PER SFRUTTARE A FONDO TUTTE LE CARATTERISTICHE
DELLA NUOVA VERSIONE DEL LINGUAGGIO DI SUN

LAVORARE CON JAVA 6

Ivan Venuti

EDIZIONI MASTER

i libri di

ioProgrammo

LAVORARE CON JAVA 6

Ivan Venuti

INDICE

Applicazioni desktop&grafica

JDBC 4.0

Scripting

Java Management Extensions

Altre novità

INSTALLAZIONE E NOVITÀ

In questo Capitolo sono presentate, suddivise per aree "tematiche", tutte le novità introdotte da Java 6. Nei prossimi capitoli gran parte di queste novità vengono analizzate nel dettaglio, insieme alle nozioni fondamentali che aiutano a comprenderne l'uso. In conclusione del Capitolo vengono dettagliati i passi necessari per il downalod e una corretta installazione del JDK.

1.1 JAVA 6: LE NOVITÀ

Java 6, chimato in precedenza "Mustag" (nome di una razza di cavalli americana) ha visto il rilascio della versione stabile il 14 dicembre 2006. Le novità presenti in questa release sono principalmente nelle seguenti aree:

- **Applicazioni Desktop e grafica:** sono state introdotte numerose caratteristiche per semplificare la creazione e l'uso delle applicazioni stand alone con pieno supporto a caratteristiche avanzate presenti nei sistemi operativi ospitanti (come l'accesso all'area delle icone, o System Tray, ma anche all'esecuzione automatica dei programmi associati a determinati tipi di file e così via); accanto a queste nuove caratteristiche, molte migliorie sono state fatte per estendere le funzionalità grafiche esistenti;
- **XML e WebServices:** sono state introdotte molte tecnologie che semplificano e migliorano (sia in termini di performance che di compatibilità con gli standard) la gestione dei file XML e la creazione di Web Services;
- **JDBC 4.0:** è stato incluso il supporto alla nuova release dello standard di accesso ai dati;
- **Scripting:** è stata attivata la possibilità di usare linguaggi di scripting all'interno delle proprie applicazioni;
- **JMX:** è stato migliorato ed esteso il supporto alle tecnologie di managment delle risorse (JMX);

- **altro:** molti altri aspetti, come l'introduzione di API del compilatore, estensione di caratteristiche per la gestione di file e interfacce di rete e così via, hanno subito modifiche e migliorie.

A ciascuna di queste aree è dedicato un Capitolo di questo libro. In conclusione verranno anche prese in considerazione le future linee di sviluppo di Java (Java 7, chiamato fin d'ora Dolphin) nonché un cenno sulla nuova licenza GPL verso cui si sta orientando Sun.
In ogni modo gran parte delle migliorie rientrano in JSR. Che si intende con questa sigla?

1.2 JCP E LE JSR

Java basa la propria evoluzione su una serie di proposte, chiamate "Java Specification Request" o, più semplicemente, JSR. In pratica quando vengono identificate delle aree di miglioramento, specifiche e focalizzate, su cui si trova un consenso sulle finalità, viene aperta una nuova JSR. All'inizio è una bozza che evolve fino a diventare una specifica vera e propria. Chi si occupa della gestione e della evoluzione delle specifiche è il Java Community Process, il cui sito di riferimento è http://jcp.org. Per semplificare il riferimento ad una determinata JSR, esse sono identificate da un numero univoco. Insieme ai documenti ancora in bozza, le JSR definitive possono essere scaricate dal sito http://jcp.org/en/jsr/detail?id=NN, dove al posto di NN si scrive il numero della specifica.

1.3 NOVITÀ IN JAVA 6: JSR 270!

Per Java 6 tutte le caratteristiche introdotte sono anch'esse parte di una JSR, la 270 (chiamata anche "Umbrella 270", inquanto raccoglie, come un ombrello, per l'appunto, le novità). Pertanto chiunque può scaricare il documento dalla pagina http://jcp.org/en/jsr/detail?id=270 e usarlo come riferimento per tutte le novità del linguaggio. Molte mi-

JSR	Descrizione
105	XML Digital-Signature APIs
173	Streaming API for XML (StAX)
181	Web-Services Metadata
199	Java Compiler API
202	Java Class-File Specification Update
221	JDBC 4.0
222	Java Architecture for XML Binding (JAXB) 2.0
223	Scripting for the Java Platform
224	Java API for XML-Based Web Services (JAX-WS) 2.0
250	Common Annotations
269	Pluggable Annotation-Processing API

Tabella 1.1 Le nuove JSR introdotte in Java 6.

gliorie sono state racchiuse in altrettante JSR; in Tabella 1.1 sono riportate tutte quelle incluse in Java 6.
Per altre JSR già introdotte sono state apportate delle modifiche (più o meno sostanziali). Le JSR interessate sono riportate in Tabella 1.2.

1.4 I DETTAGLI

La già ricordata JSR 270 elenca tutte le nuove caratteristiche di Java 6. In tale documento esse sono (anche) ordinate per priorità (ovvero per importanza). Quelle più importanti sono dette Release Drivers (RD, ovvero quelle la cui presenza è, per importanza pratica o "strategica", essenziale nella nuova release), le Group Drivers (o semplicemente GD) sono considerate importanti ma la loro importanza è

JSR	Descrizione
3	Java Management Extensions (JMX)
160	Java Management Extensions (JMX) Remote API 1.0
166	Gestione della concorrenza
206	Java API for XML Processing 1.4

Tabella 1.2 Le JSR che in Java 6 sono state migliorate.

comunque minore delle RD e, infine le Target of Opportunity (TO), ovvero caratteristiche che sono state incluse in quanto erano una buona opportunità per farlo (ma che, quando Java 6 non era ancora stato rilasciato, non avevano un impatto così determinante sulla release da rimandarne il rilascio; se non erano pronte quando lo erano tutte le altre caratteristiche, si attendeva una futura release per introdurle). In pratica senza una RD la release non sarebbe uscita; ogni sforzo è stato fatto per includere le GD, le TO ci sono perché si è fatto in tempo a completarle quando il resto erano disponibili.

1.5 RELEASE DRIVERS

In questa categoria ci sono gran parte delle JSR previste; in particolare:

JSR 202: Java Class-File Specification Update
JSR 105: XML Digital-Signature APIs
JSR 199: Java Compiler API
JSR 269: Pluggable Annotation-Processing API
JSR 221: JDBC 4.0
JSR 173: Streaming API for XML (StAX)
JSR 181: Web-Services Metadata
JSR 222: Java Architecture for XML Binding (JAXB)
JSR 224: Java API for XML-Based Web Services (JAX-WS)
JavaBeans Activation Framework (JAF) 1.

Di esse si parlerà diffusamente anche nel prosieguo del libro analizzando esempi d'uso e indicando, dove possibile, articoli e risorse per approfondirle.

1.6 GROUP DRIVERS (GD)

In questa categoria sono state classificate ben due JSR che, almeno

in parte, potrebbero sorprendere visto che comunque è stata data ampia risonanza e molti nutrivano forti attese su di esse:

JSR 223: Scripting for the Java Platform
JSR 250: Common Annotations

1.7 IMPORTANTI NOVITÀ CHE RIGUARDANO LA TECNOLOGIA JMX:

Descrittori generalizzati per gli MBean: grazie ad essi è possibile descrivere l'MBean stesso e non solo i suoi componenti;
MXBeans: un nuovo modello di MBean (presente in Java 5 tra gli MBean di sistema ma in cui non era possibile crearne di nuovi); con Java i programmatori possono sfruttare le loro caratteristiche semplificate; i dettagli verranno mostrati nel seguito.
Anche queste novità verranno approfondite e analizzate con esempi concreti. Molte nuove caratteristiche per una miglior gestione delle applicazioni e aggiornamenti alle librerie grafiche trovano collocazione nelle GD:

- Supporto per la gestione delle System-tray native (purché l'interfaccia grafica del sistema ospite le supporti; Vista, Windows XP e i maggiori desktop manager per Linux lo fanno!);
- Semplici meccanismi per l'ordinamento, il filtraggio e la selezione di elementi in una JTable (prima questa operazione era possibile solo ricorrendo a tool di terze parti o scrivendo propri componenti, ma la cui realizzazione era comunque complessa);
- Attach-on-demand: mentre prima (Java 5) era possibile collegare un monitor JMX o qualche altro tool di profilazione solo se la JVM era stata inizializzata con opportuni parametri di avvio, ora è possibile farlo con qualsiasi JVM senza particolari accorgimenti;

- Miglioramenti sulla classe java.io.File: ora è possibile conoscere lo spazio totale, libero e utilizzabile su file system; inoltre sono state aggiunte funzionalità per la gestione dei diritti d'accesso ai singoli file;
- Aggiornamenti alle librerie java.util.concurrent (introdotte in Java 5);
- Password prompting per le librerie di I/O da console (mentre prima questa caratteristica era presente solo per interfacce grafiche);
- Pluggable locale data: è possibile usare nuove localizzazioni per il JRE in maniera dinamica (ovvero aggiungere supporto per un'ulteriore lingua "a caldo"), grazie ad una architettura a plug-in.

1.8 TARGETS OF OPPORTUNITY (TO)

Infine ecco una serie di caratteristiche di cui è stata decisa l'inclusione (e che verranno analizzate successivamente) e classificate come TO:

- Accesso alle applicazioni desktop con le applicazioni native per la gestione di URL, indirizzi di email e mime type associati a specifiche applicazioni (come PDF ad Acrobat Reader, .doc a Microsoft Word e così via);
- Gestione di splash screens: per creare immagini di benvenuto che vengano visualizzate velocemente, senza attendere che l'engine grafico sia stato inizializzato;
- Nuove finestre di dialogo modali: permettono una gestione più raffinata rispetto alle due uniche opzioni precedenti (modale/non modale);
- Accesso ai parametri di rete: questo permette di indagare e conoscere le capacità di connessione e le diverse configurazioni dei dispositivi di rete presenti sulla macchina dove viene eseguito il programma.

Ecco un'altra serie di miglioramenti classificati come TO; per essi si farà un breve cenno in quanto non verranno affrontate ulteriormente nel resto del libro:

- Deques come nuova struttura dati (è una lista o una coda ma in cui si può accedere ad entrambe le estremità);
- Sorted sets e maps con navigazione nei due sensi e con primitive per il reperimento di elementi che soddifano criteri di confronto come "minore di" e "maggiore di";
- Nuovo costrutto SwingWorker per semplificare l'uso di thread nelle interfacce grafiche;
- GIF image writer: finalmente è possibile usare anche il formato GIF per l'esportazione di immagini; la sua assenza era dovuta unicamente a questioni legate al copyright sull'algoritmo alla base del formato GIF (copyright che è scaduto e quindi non più limitante);
- Miglioramenti alla gestione dei Resource-bundle: essi sono il meccanismo standard per l'internazionalizzazione delle applicazioni Java; grazie a questi miglioramenti è possibile aggiornare dinamicamente un resource bundle già caricato (permettendo cioè di toglierlo dalla cache e di sostituirlo con una nuova versione) e possibilità di definire strategie di risoluzione personalizzate; è anche previsto un nuovo formato XML per la loro rappresentazione;
- Normalizzazione di stringhe Unicode: introdotte nuove funzionalità di normalizzazione da altri set di caratteri;
- Baseline/gap API: permette la creazione di layout grafici non legati ad un determinato look&feel; inoltre forniscono l'accesso a informazioni quali distanze preferite tra componenti grafici (gaps) e determinazione della base del testo (baseline) in maniera da allinearvi correttamente altri componenti grafici;
- Semplificazione dell'uso dei layout manager in Swing, grazie all'introduzione del più intuitivo GroupLayout;

- Migliorato il meccanismo di drag&drop nei componenti Swing;
- JTabbedPane: migliorato l'uso dei Tabs affinché possano contenere componenti arbitrari (si pensi ai nuovi tab di navigazione di Mozilla FireFox e Internet Explorer 7);
- Text-component printing: è stato migliorato il supporto alla stampa della classe JTextComponent affinché il suo contenuto possa essere stampato come un documento;
- La JVM permette l'accesso alle informazioni dello heap (indispensabile per il debug);
- La JVM permette un insieme multiplo (e simultaneo) di agents (tecnologia JMX)
- Array reallocation: è stata estesa la classe java.util.Arrays con due nuovi metodi (copyOf e copyOfRange) per la riallocazione di array statici;
- Floating point: aggiunte funzioni raccomandate dallo standard IEEE 754 (nei tipi di dato floating point predefiniti e nei relativi wrapper);
- Generalized lock monitoring: in Java 5 sono state introdotte le classi della gerarchia java.util.concurrent ma senza fornire meccanismo per il monitoring di possibili lock su tali classi; Java 6 introduce primitive per farlo e per gestirli;
- Annotations generiche per la descrizione degli MBean: come per altre caratteristiche, l'uso delle annotazioni semplifica la scrittura del codice sorgente; è compito dei tool del JDK quello di associare queste descrizioni secondo il nuovo meccanismo dei descrittori generalizzati per MBean;
- Supporto ai nomi di dominio internazionalizzati: secondo le RFC 3490, 3491, e 3492 i nomi di dominio possono contenere caratteri non ASCII; Java 6 ora ne permette la corretta gestione;
- Reso disponibile un semplice HTTP cookie manager: mentre in precedenza (Java 5) era stata definita un'apposita interfaccia per la gestione dei cookie, in questa release è fornito anche un semplice manager che ne implementa le funzionalità;

- Meccanismo per il lookup di Service-provider: grazie ad una nuova classe, java.util.ServiceLoader, è possibile intervenite da programma per accedere ai service provider (meccanismo introdotto nei JAR dalla versione J2SE 1.3).

1.9 PREPARARE L'AMBIENTE DI LAVORO

Prima di partire è bene predisporre un ambiente, installando e configurando opportunamente Java 6, per provare gli esempi proposti.

1.10 INSTALLARE IL JDK

Il JDK "ufficale" può essere scricato dal sito della Sun; il link è disponibile, insieme alle altre versioni della piattaforma, alla pagina http://java.sun.com/javase/downloads/index.jsp (**Figura 1.1**).
Prima di eseguire il download bisogna accettare le condizioni d'uso. Per il download si consiglia di installare il download manager fornito dalla Sun che permette, tra le altre cose, il ripristino di download interrotti. Si è parlato di JDK "ufficiale" in quanto altre aziende potrebbero fornire JDK compatibili. Un esempio è quello fornito dalla

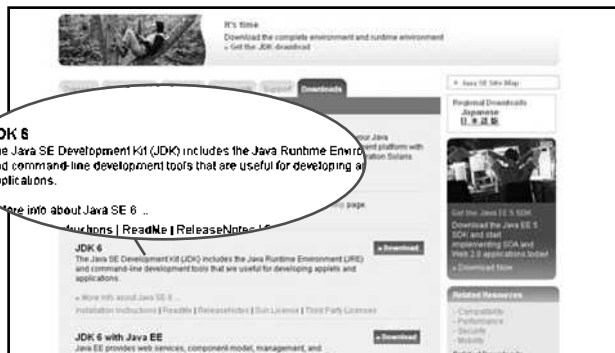



Figura 1.1: Pagine per il downloaddel JDK

IBM (si veda la pagina http://www.ibm.com/developerworks/java/jdk/). In linea di principio ogni JDK deve soddisfare caratteristiche standard e, pertanto, essere compatibile, come linguaggio, con quello ufficiale; le diversità possono esserci nei tool di sviluppo a corredo e alle performance dell'interprete/compilatore. Questo libro farà riferimeto al JDK della Sun e saranno descritti alcuni dei tool ivi presenti. Effettuato il download esso va eseguito e installato in una cartella a propria scelta.

1.11 VARIABILI D'AMBIENTE

Una volta installato il JDK è opportuno settare opportunamente alcune variabili d'ambiente; benché questa operazione non sia sempre fondamentale, è vivamente consigliata sia perché in questo modo è più comodo utilizzare gli strumenti del JDK sia perché molte applicazioni, che si basano su Java, fanno uso di alcune di queste informazioni. Per conoscere le variabili d'ambiente già presenti bisogna aprire una console di comandi. In Windows va digitato:

```
set
```

in Linux, invece, si digiti il comando:

```
env
```

In entrambi i casi vengono stampate a video tutte le variabili d'ambiente già definite. Per agevolare lo sviluppo di programmi Java si definiscano le variabili d'ambiente JAVA_HOME e CLASS_PATH in maniera che la prima indichi la cartella di installazione del JDK, la seconda dove trovare le classi (inizialmente i due valori coincideranno); e si provveda anche a modificare la variabile PATH affinché vengano aggiunti gli eseguibili del JDK. In **Tabella 1.3** le variabili da definire per i sistemi Windows e in **Tabella 1.4** quelle per Linux, con la descrizio-

Variabile	Valore (consigliato)	Descrizione
JAVA_HOME	set JAVA_HOME=c:\cartella\path\	Cartella di installazione del JDK
CLASSPATH	set CLASSPATH=.;%JAVA_HOME%	Sia la cartella di installazione del JDK che quella corrente (indicata con il carattere "punto": "." e che è consigliabile mettere per prima)
PATH	set PATH=%PATH%;%CLASSPATH%\bin	Mantenere il PATH definito di default dal sistema e aggiungervi anche quello della cartella /bin del JDK

Tabella 1.3 Variabili d'ambiente da settare (Windows); si noti l'uso del carattere ";" per separare una lista di valori e di %nome% per riferirsi ad una variabile definita in precedenza.

Variabile	Valore (consigliato)	Descrizione
JAVA_HOME	export JAVA_HOME=/cartella/path	Cartella di installazione del JDK
CLASSPATH	export CLASSPATH=.:$JAVA_HOME	Sia la cartella di installazione del JDK che quella corrente (indicata con il carattere "punto": ".").
PATH	export PATH=$PATH:$JAVA_HOME/bin	Mantenere il PATH definito di default dal sistema e aggiungervi anche quello della cartella /bin del JDK

Tabella 1.4 Variabili d'ambiente da settare (Linux); si noti l'uso del carattere ":" per separare una lista di valori e di $nome per riferirsi ad una variabile definita in precedenza.

ne del loro significato e i valori consigliati. Le variabili sono definibili direttamente da una console di comandi ma hanno un valore locale ad essa (in pratica vengono perse quando si chiude la console corrente e, similmente, aprendo nuove console le variabili definite nella prima non sono propagate alle altre). Spesso, invece, è comodo definirle una volta per tutte. In Windows Xp le variabili d'ambiente si definiscono, in maniera permanente, ovvero che persiste anche nelle successive sessioni, facendo clic con il pulsante destro su "Risorse del sistema", scegliendo "Proprietà" dal menu contestuale. Dalla finestra si selezioni il tab "Avanzate" e quindi il pulsante "Variabili d'ambiente". In Linux invece è necessario editare il file di configurazione della shell in uso. Se essa, com'è di solito, è la bash, bisognerà editare il file .bash-profile presente nella cartella home dell'utente con cui ci si collega (tale cartella è quella che si accede eseguendo il comando "cd" senza parametri). Si faccia attenzione anche al fatto che in Linux tutti i comandi e i nomi di variabili di sistema sono "case sensitive", ovvero è importante rispettare le maiuscole/minuscole dei nomi; in Windows, invece, maiuscolo o minuscolo è lo stesso (in questo caso si dice che le variabili, e i loro valori, sono "case insensitive").

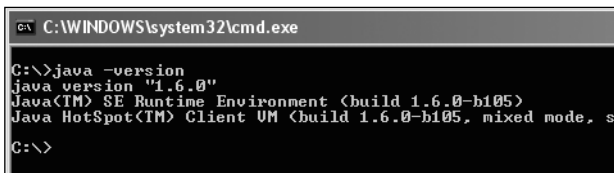
1.12 SIGNIFICATO DELLE VARIABILI D'AMBIENTE

La variabile d'ambiente PATH indica al sistema operativo dove si trovano i comandi eseguibili. Infatti nei sistemi operativi parte dei comandi eseguibili (o tutti) sono salvati sul file system e possono essere esegui-

Attenzione

Se si vuol visualizzare il valore di una variabile d'ambiente è necessario aprire una console di comandi e digitare il comando echo %nomevariabile% in Windows, mentre il comando echo $nomevariabile è utilizzabile sui sistemi Linux.

ti o indicando il nome completo di percorso (per esempio c:\cartella\fileeseguibile.exe, su Windows, o /usr/bin/comando su Linux) oppure indicando solo il loro nome (come "fileeseguibile.exe" o "comando") e lasciando al sistema operativo il compito di cercare, sul disco, la posizione dove esso si trova. È indubbio che, per i comandi d'uso comune, quest'ultima strada è da preferirsi alla prima (sia perché permette di scrivere meno caratteri sia perché ci si può dimenticare il percorso sul disco dove si trovano i diversi comandi). D'altro canto è improponibile che il sistema operativo debba ricercare su tutto il disco un eventuale comando (normalmente ci sono centinaia di cartelle contenenti migliaia di file: la ricerca su tutto il disco impiegherebbe ogni volta alcuni minuti!). Per questo motivo alcune variabili d'ambiente servono proprio a indicare dove si trovano i comandi da eseguire. Se, all'interno della variabile d'ambiente, è specificato anche un punto (".") allora la ricerca avviene anche nella cartella dove ci si trova nel momento in cui si digita il comando. In maniera molto simile a PATH (usata per specificare i comandi) c'è la variabile CLASSPATH che indica dove reperire le classi e le librerie specifiche di Java.

1.13 TEST DELL'AMBIENTE

Una volta definito le variabili sopraccitate, si apra una console e si provi a digitare:

```
java -version
```

se tutto è andato a buon fine dovrebbe apparire la versione dell'interprete Java (versione che coincide con quella del JDK installato.18

1.14 EDITARE, COMPILARE ED ESEGUIRE

Un programma Java può essere scritto con un qualsiasi editor di te-

```
C:\WINDOWS\system32\cmd.exe

C:\>java -version
java version "1.6.0"
Java(TM) SE Runtime Environment (build 1.6.0-b105)
Java HotSpot(TM) Client VM (build 1.6.0-b105, mixed mode, s

C:\>
```

Figura 1.2: Verifica che l'ambiente sia configurato correttamente

sto (come NotePad per Windows o vi per Linux). Una volta scritto il programma va compilato. Per compilarlo si può usare il compilatore del JDK:

```
javac nomeFile.java
```

Se tutto è andato a buon fine si ottiene il file compilato che ha lo stesso nome del file sorgente ma con estensione .class. Quest'ultimo può essere eseguito usando un interprete:

```
java nomeFile
```

Evidentemente è noioso editare, compilare ed eseguire ogni singola classe. Per questo è consigliato ricorrere a dei tool, chiamati IDE (Integrated Development Enviroment) dove queste operazioni sono semplificate e ottenibili tutte da comandi del tool. Non solo: sono disponibili funzionalità evolute quali l'autocompletamento del codice, funzionalità di debug, analisi del codice e così via.
Quale IDE? NetBeans, per esempio
Tra i primi tool completamente compatibili con il JDK 1.6 e liberamente utilizzabili, si segnala NetBeans (l'importante è avere una versione 5.5 o successiva). Per verificare cheil JDK utilizzato sia proprio l'1.6, si apra NetBeans e si scelga "Tools > Java Platform Manager". Nella finestra che si apre compaiono i diversi JDK riconosciuti. È importante che ci sia anche l'1.6
Se il JDK 1.6 non dovesse comparire, usare il pulsante "Add platform…"

per una ricerca manuale sul file system della cartella in cui tale JDK si trova.

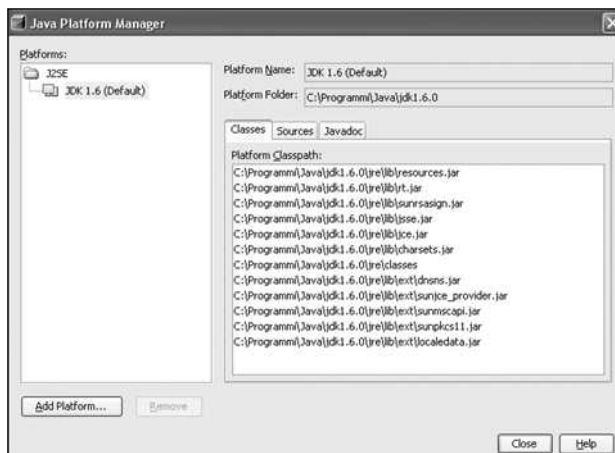



Figura 1.3: Riconoscimento del JDK 1.6 da parte di NetBeans

1.15 UN AMBIENTE CON TANTI "SERVER"!

È interessante osservare che il JDK 6.0 prevede alcuni server integrati direttamente nel JDK. In particolare è presente un server http per provare le applicazioni e un server DB per l'accesso alle basi dati. Come server DB è stato scelto Apache Derby (sito di riferimento http://db.apache.org/derby/), ma che in questo contesto è stato ri-

Attenzione

Nel proseguio del libro alcuni esempi faranno riferimento a NetBeans come ambiente di sviluppo. Anche i sorgenti del libro sono distribuiti come progetto di NetBeans. Quanto descritto può essere usato, in maniera analoga, con un qualsiasi altro editor con supporto del JDK 1.6 o, volendo, usando i tool a linea di comando forniti di default con il JDK.

denominato JavaDB; come server http è stato scelto Tomcat (http://tomcat.apache.org). In seguito verranno forniti i dettagli sul loro uso (in particolare quando saranno analizzate le funzionalità per la costruzione di Web Services e quando si approfondirà l'accesso alle basi dati con JDBC 4.0).

APPLICAZIONI DESKTOP & GRAFICA

Una delle critiche che spesso vengono rivolte a Java è la sua scarsa integrazione con gli ambienti grafici (detti anche gestori di finestre) su cui vengono eseguite le sue applicazioni. In Java 6 sono presente una serie di novità su questo fronte che, seppur non "epocali", di sicuro aggiungono caratteristiche interessanti e di cui si sentiva la mancanza. Accanto a questi aspetti alcuni miglioramenti riguardano la creazione e gestione di GUI e altre caratteristiche legate alle primitive grafiche (anche le performance sono state notevolmente migliorate).

2.1 APPLICAZIONI STANDARD

Qualunque sistema operativo permette di associare ad alcuni tipi di file un'applicazione predefinita per un certo tipo di azione (infatti, facendo doppio clic su un file PDF viene mandato in esecuzione Acrobat Reader, se questo è installato e correttamente configurato; così la maggior parte delle usuali estensioni possiede un programma adatto alla loro gestione). Allo stesso modo, se in una pagina Web si fa clic su un indirizzo di posta elettronica, viene aperto il client di posta predefinito. Queste, e altre, operazioni ora sono possibili anche da applicazioni Java, grazie alla nuova classe java.awt.Desktop.
Prima di far uso di una sua qualche funzionalità è necessario verificare se effettivamente l'ambiente in cui viene esguito il programma le supporta; per farlo cè un metodo apposito:

```
import java.awt.Desktop;
...
if ( Desktop.isDesktopSupported() ){
  // si può usare le sue funzionalità
}
```

Se non si effettua tale controllo, un'invocazione dei metodi che

verranno descritti fra poco porterà ad una eccezione al run-time. Dalla classe Desktop si può invocare il metodo getDesktop() per reperire un oggetto (che è sempre lo stesso per tutte le applicazioni, secondo il design pattern singleton) su cui invocare le azioni standard per i diversi oggetti:

```
Desktop desktop = Desktop.getDesktop();
```

nel seguito si assumerà che esistano sul file system due file nella cartella principale: c:\test.txt (file di solo testo) e c:\file.htm (un file contenente un qualsiasi documento HTML) su cui verranno eseguiti semplici test.

2.2 LE AZIONI PREVISTE

Dall'oggetto di tipo Desktop è possibile eseguire una delle azioni specificate da Desktop.Action: browse, edit, mail, open e print. Come si può intuire, esse corrispondono alla navigazione, editazione, invio di email, apertura e stampa. Per ciascuna azione è possibile verificare se il sistema operativo la supporta (da notare che il fatto che essa venga supportata non significa automaticamente che l'operazione vada a buon fine: si pensi al caso di stampa di un file che non esiste o all'apertura su un tipo di file per cui non è associata alcuna applicazione!).
Come usare le diverse azioni previste dalle API? Il metodo browse accetta come argomento un oggetto di tipo java.net.URI; ecco come applicarlo ai due file di esempio:

```
URI uriTxt = new URI("file://c:/test.txt");
URI uriHtml = new URI("file://c:/test.htm");
if (desktop.isSupported( Desktop.Action.BROWSE )){
  desktop.browse(uriTxt);
  desktop.browse(uriHtml);
```

```
}else
  System.out.println("browse non supportata");
```

Usando l'azione browse si può osservare che i file vengono aperti, rispettivamente, con l'editor predefinito (nel sistema in cui è stato fatto il test NotePad) e il Web Browser predefinito (in questo caso Mozilla Firefox) ciascuno in una sua finestra (**Figura 2.1**).

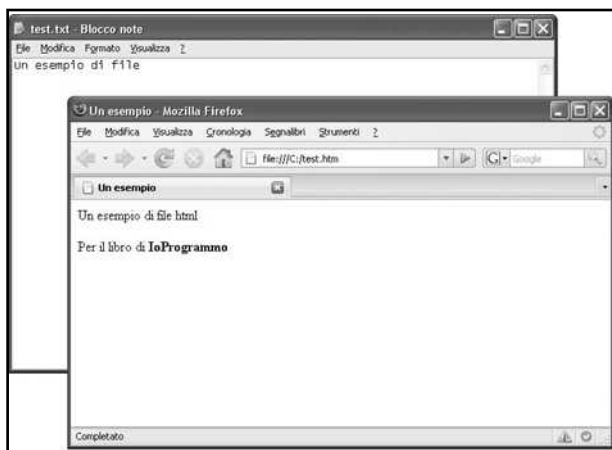



Figura 2.1: il metodo browse sui due file di esempio (uno testuale l'altro html).

L'azione sucessiva è quella di editing; a differenza della precedenza è necessrio fornire al metodo un oggetto di tipo java.io.File (in effetti browse si applica a risorse comunque disponibili in rete, l'editing per risorse su file system):

```
File fileTxt = new File("c:/test.txt");
File fileHtml = new File("c:/test.htm");
if (desktop.isSupported( Desktop.Action.EDIT )){
  desktop.edit(fileTxt);
```

```
  desktop.edit(fileHtml);
}else
  System.out.println("edit non supportata");
```

Il file testuale continua a venire aperto con l'editor predefinito; il file html, invece, con l'editor definito per i file html (sul sistema di test è HTML-kit, **Figura 2.2**).

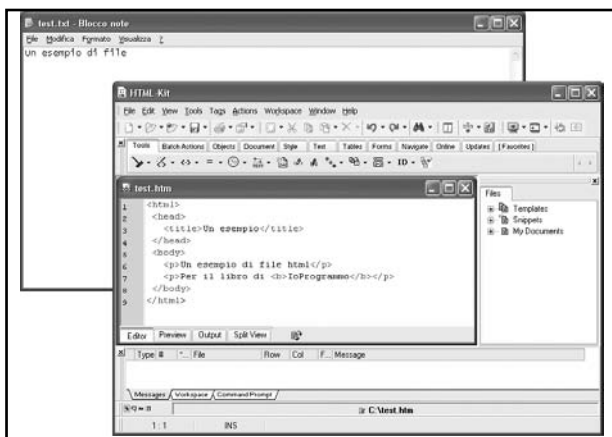



Figura 2.2: il metodo browse sui due file di esempio (uno testuale l'altro html).

Il metodo successivo che si analizza è open:

```
if (desktop.isSupported( Desktop.Action.OPEN )){
  desktop.open(fileTxt);
  desktop.open(fileHtml);
}else
  System.out.println("open non supportata");
```

Nell'esempio specifico il comportamento è analogo all'uso di browse (**Figura 2.1**); si noti però come browse accetti un parametro di

tipo URI, open, invece, un parametro di tipo File. Esistono due metodi mail: uno senza parametri, l'altro con un parametro di tipo URI. Invocando il primo metodo viene aperta una mail vuota:

```
if (desktop.isSupported( Desktop.Action.MAIL ))
  desktop.mail();
else
  System.out.println("mail non supportata");
```

invocando il secondo una mail il cui destinatario è impostato con l'indirizzo passato come argomento:

```
URI email=new URI("mailto:ivanvenuti@yahoo.it");
if (desktop.isSupported( Desktop.Action.MAIL ))
  desktop.mail(email);
else
  System.out.println("mail non supportata");
```

L'ultimo metodo, print, permette di mandare in stampa il file passato come argomento:

```
if (desktop.isSupported( Desktop.Action.PRINT ))
  desktop.print(fileTxt);
else
  System.out.println("print non supportata");
```

2.3 ACCESSO ALL'AREA DELLE ICONE

Alcuni gestori di finestre mettono a disposizione delle applicazioni un'area particolare dove è possibile mettere delle icone. In Windows, per esempio, quest'area è sulla barra delle applicazioni in basso a destra (Figura 2.3); al clic su una delle icone viene eseguita un'azione standard per quell'applicazione. Facendo clic con il pulsante destro

può apparire un menu contestuale da dove è possibile scegliere una tra le azioni contemplate per l'applicazione. Java 6 mette a disposizione una serie di oggetti per poter interagire con quest'area del sistema. In particolare l'oggetto java.awt.SystemTray permette di restituire un riferimento a tale area, mentre istanziando nuovi oggetti di tipo java.awt.TrayIcon è possibile creare nuove icone e personalizzarne il comportamento.

Figura 2.3: la system tray di Windows.

2.4 UN THREAD DI ESEMPIO

Per illustrare l'uso della gestione della SystemTray, viene creato un thread che si occuperà della visualizzazione di una nuova icona e della gestione delle azioni su di essa. Un thread è un programma che, una volta eseguito, resta attivo in memoria ma restituisce il controllo al chiamante. Ci sono vari modi, in Java, di realizzarlo. Si userà il seguente: si definisce una nuova classe che implementa l'interfaccia Runnable, la quale prevede la creazione di un metodo run.

```
public class TestSystemTray implements Runnable {
  public void run(){
  }
}
```

All'interno del metodo run verrà scritta tutta la logica per la creazione di una nuova icona nell'apposita area a loro dedicata. Per mandare in esecuzione il thread basta istanziare un nuovo oggetto (con l'usuale costruttore) per la classe TestSystemTray: in automatico verrà chiamato il metodo run dopo la sua creazione. Ecco, per esempio, come settare il look&feel specifico per il sistema su cui viene eseguita l'applicazione e viene mandato in esecuzione il thread:

```
public static void main(String[] args) {
 try {
   UIManager.setLookAndFeel(
     UIManager.getSystemLookAndFeelClassName());
 } catch (Exception ex) {
   ex.printStackTrace();
 }
 EventQueue.invokeLater(new TestSystemTray());
}
```

Come fatto per altri metodi, anche questa classe è dotata di un metodo che permette di sapere se il sistema operativo sottostante gestisce la SystemTray; se la supporta, c'è anche un metodo per ottenere il riferimento all'oggetto (sempre secondo il pattern singleton) che la gestisce:

```
public void run(){
   if (SystemTray.isSupported()){
```

Attenzione

Si noti l'uso di invokeLater. Infatti, essendo questo un thread che disegna un'interfaccia grafica, non deve interferire con il thread principale, dedicato alla creazione dell'interfaccia grafica stessa. Per risolvere questo e altri problemi Java 6 introduce l'oggetto SwingWorker.

```
        SystemTray sysTray = SystemTray.getSystemTray();
        // fai qualcosa...
    } else {
        System.out.println("Il sistema non supporto il SystemTray");
    }
}
```

Ora la creazione di una nuova icona prevede la creazione di un oggetto TrayIcon; il costruttore della classe prende tre parametri: il primo è l'icona che verrà visualizzata nella barra, il secondo è il messaggio mostrato quando l'utente posiziona il mouse sull'icona, il terzo (e ultimo) parametro prevede la creazione di un menu che apparirà quando l'utente fa clic sull'icona con il pulsante destro del mouse:

```
final Image icon =
  Toolkit.getDefaultToolkit().getImage("c:/icona.gif");
final TrayIcon trayIcon = new TrayIcon(
  icon,
 "Java6: esempio d'uso di SystemTray",
 getMenu()
);
```

Nell'esempio è stata creata un'icona di dimensione 14x14 in formato gif. Il metodo getMenu(), che effettivamente si occupa di costruire il menu (di tipo PopupMenu), lo si illusterà in seguito. Avendo a disposizione un oggetto di tipo TrayIcon, ecco come aggiungerlo al system tray e mostrare un messaggio (il messaggio è simile ai messaggi che si vede apparire quando si inserisce una penna USB nel PC):

```
try {
  SystemTray.getSystemTray().add(trayIcon);
```

```
  trayIcon.displayMessage(
    "Java6, uso di SystemTray",
    "Usare il pulsante destro per le opzioni",
    TrayIcon.MessageType.INFO);
} catch (AWTException ex) {
  ex.printStackTrace();
}
```

In **Figura 2.4** si può osservare il risultato dopo aver mandato in esecuzione il programma di esempio.
Non resta che vedere come potrebbe essere realizzato il menu da mostrare sul clic destro del mouse; ecco come creare tale menu con tre opzioni e una suddivisione dopo le prime due:

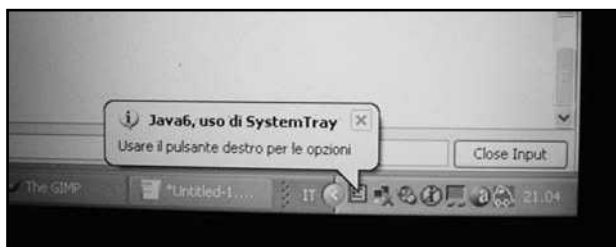



Figura 2.4: L'icona aggiunta e il messaggio mostrato

```
private PopupMenu getMenu(){
  PopupMenu menu = new PopupMenu();

  MenuItem javaMenu = new MenuItem("Enviroment");
  javaMenu.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
     JOptionPane.showMessageDialog(null, DefSistema());
    }
  });

```

```
 MenuItem aboutMenu = new MenuItem("About");
 aboutMenu.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
    JOptionPane.showMessageDialog(null,
      "Libro sulle novità in Java 6,\n"+
       "Autore: Ivan Venuti\n"+
       "Sito Web: http://ivenuti.altervista.org");
    }
  });

  MenuItem exitMenu = new MenuItem("Esci");
  exitMenu.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
   });
  menu.add(javaMenu);
  menu.add(aboutMenu);
  menu.addSeparator();
  menu.add(exitMenu);

  return menu;
}
```

La prima opzione mostra le entry definite nell'ambiente di esecuzione, la seconda mostra un semplice messaggio di cos'è l'applicazione (about) il terzo pulsante termina l'esecuzione del thread (e quindi permette di eliminare la TraiIcon dalla System Tray).
In **Figura 2.5** il menu visualizzato in seguito al clic destro del mouse sull'icona. In **Figura 2.6** la finestra con le variabili definite nell'ambiente di test (mostrate scegliendo la prima opzione del menu).
Ecco i dettagli del metodo che si preoccupa di reperire le variabili e di costruire un'apposita stringa pronta per essere visualizzata:

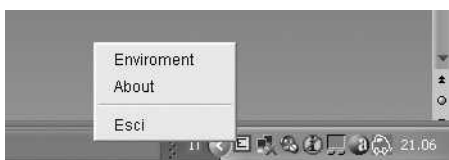



Figura 2.5: Il menu contestuale visualizzato dopo un clic con il pulsante destro sull'icona

```
private String DefSistema(){
 java.util.Map<String,String>

   env = System.getenv();
 StringBuffer tmp = new StringBuffer();

 for (String elem : env.keySet()) {
   String tmpStr = env.get(elem);
   if (tmpStr!=null && tmpStr.length()>70)

     tmpStr = tmpStr.substring(0, 70)+" [...]";
   tmp.append(elem).append(": ").
     append(tmpStr).append("\n");
 }
 return tmp.toString();
}
```

2.5 MIGLIORAMENTI ALLA GESTIONE GRAFICA

Di seguito alcuni tra i tanti aspetti che sono stati migliorati per la creazione di interfacce grafiche. C'è anche da segnalare un miglior supporto delle primitive di sistema da parte delle routine grafiche; questo ha portato a sensibili miglioramenti di performance.

Figura 2.6: Il messaggio mostrato selezionando "Enviroment" dal menu contestuale

2.6 UNO SPLASH SCREEN

La creazione degli splash screen (quelle immagini mostrate durante l'avvio di un'applicazione) è sempre stato problematico in Java: il motivo è che prima di poter visualizzare l'immagine grafica, l'intero ambiente grafico deve essere opportunamente predisposto, causando un ritardo nella visualizzazione che, ovviamente, fa diminuire la sua utilità (infatti ha senso usare lo splash screen proprio per mostrare velocemente alcune informazioni nell'attesa di caricare il resto del programma!). Con Java 6 è stato superato questo problema. La prima soluzione consiste nell'invocare opportunamente l'interprete, specificando quale immagine mostrare come splash screen in attesa dell'avvio dell'applicazione:

```
>java –splash:nomeImmagine.jpg nomeProgramma
```

Si noti che funziona anche l'esecuzione di java specificando la sola immagine:

```
>java –splash:nomeImmagine.jpg
```

In pratica poco prima di mostrare le opzioni disponibili per l'interprete, viene mostrata l'immagine specificata. Ovviamente l'esecuzione dell'output è così veloce che la presenza dello splash screen è quasi impercettibile (ma c'è!). Per provare gli splash screen si farà uso di questa semplice classe, in cui, nel costruttore, viene invocato il metodo Thread.sleep per fare una pausa di 3 secondi:

```
package it.ioprogrammo.java6.desktop;

public class TestSplash {

  public TestSplash() {
    try {

      Thread.sleep(3000);
    } catch (InterruptedException ex) {
      ex.printStackTrace();
    }
  }

  public static final void main(String[] args){

    TestSplash ts = new TestSplash();
  }
}
```

C'è una seconda e ultima possibilità per creare uno splash screen; essa si applica al caso di programmi distribuiti con file JAR. Nel file META-INF/manifest.mf, che contiene una serie di proprietà dell'archivio compresso, è stata aggiunta una voce, chiamata SplashScreen-Image. Assegnando a questa nuova proprietà un nome di immagine, quest'ultima viene usata come splash screen all'esecuzione dell'applicazione.

Attenzione
È possibile sia specificare l'immagine da usare sia a linea di comando che, contemporaneamente, nel file META-INF/manifest.mf; in caso che le immagini siano diverse ha precedenza quella indicata sulla linea di comando.

2.7 GESTIRE LO SPLASH SCREEN

In Java 6 non c'è la possibilità di creare uno splash screen se questo non è stato specificato né da linea di comando né nel file manifest.mf. Però se ne viene specificato almeno uno allora si ha a disposizione un oggetto ad hoc per gestirlo da programma: java.awt.SplashScreen. Ecco, per esempio, come personalizzare l'immagine aggiungendo delle scritte (il risultato è mostrato in **Figura 2.7**):

```
public static final void main(String[] args){
  SplashScreen splash = SplashScreen.getSplashScreen();
  try {

    if (splash!=null){
       Graphics2D g = splash.createGraphics();
       g.setFont( new Font(Font.SANS_SERIF, Font.ITALIC, 38) );
       g.drawString("Libro Java 6", 15, 30);
       g.setFont( new Font(Font.SANS_SERIF, Font.PLAIN, 12) );
       g.drawString("di Ivan Venuti", 30, 50);
       splash.update();
    }
  } catch (Exception ex) {
    ex.printStackTrace();

  }
  TestSplash ts = new TestSplash();
```

```
  if (splash!=null)
    splash.close();
}
```

Figura 2.7: Lo splash screen con scritte personalizzate

Si presti attenzione che le scritte appaiono solo al momento dell'invocazione del metodo update; inoltre è possibile decidere quando non visualizzare più lo splash screen invocando il suo metodo close. Nulla vieta di scrivere le scritte (o altri componenti grafici) in maniera interattiva man mano che i diversi componenti sono caricati durante lo startup dell'applicazione, rendendo esplicito all'utente lo stato di avanzamento dello stesso (un po' come avviene per lo splash screen di NetBeans: se si fa caso c'è una piccola barra di stato e vengono stampate, via via, le componenti caricate).

2.8 FINESTRE MODALI: NUOVE CARATTERISTICHE

Prima di Java 6 le finestre che presentavano finestre di dialogo ("dialog box") potevano essere modali oppure no: nel primo caso l'utente era obbligato a interagire con la dialog box prima di poter procede, nel secondo caso potevano anche ignorare la dialog box e la-

sciarla in backgroud. Il problema di una siffatta gestione era che il comportamento modale/non modale era troppo restrittivo per alcuni casi particolari. Si pensi al caso in cui sia visibile una dialog box modale ma che si vorrebbe poter accedere ad una funzionalità di help in linea. Questo non era possibile in quanto la dialog box "esigeva" una qualunque azione prima di procedere. Ora, con Java 6, ci sono a disposizione ben quattro comportamenti modali:

Modeless: in questo caso non c'è alcun comportamento modale (presente anche prima di Java6);
Document-Modal: blocca tutte le finestre dello stesso documento; per documento si intende la gerarchia di finestre che hanno il padre in comune con il dialog box a cui si applica questa modalità;
Application-Modal: blocca tutte le finestre della stessa applicazione (si faccia attenzione che nel caso di più applet è il browser che decide se trattarle come applicazioni separate o come un'unica applicazione);
Toolkit-Modal: blocca tutte le finestre che sono eseguite dal medesimo toolkit (presente anche prima di Java 6: era questo il comportamento modale quando veniva specificato). La differenza rispetto ad Application-Modal sta solo per Applet e applicazione eseguite via Java WebStart.

Per far uso dei nuovi comportamenti è stato introdotto il seguente costruttore:

Attenzione

Dove l'owner è il componente Window che è padre del nuovo dialog box; se si passa null, significa che a divenirne padre è che un frame condiviso creato automaticamente (ma che resta invisibile e, per questo, è come se non ci fosse). Nel caso si passi null, la modalità document-modal è equivalente a quella modeless.

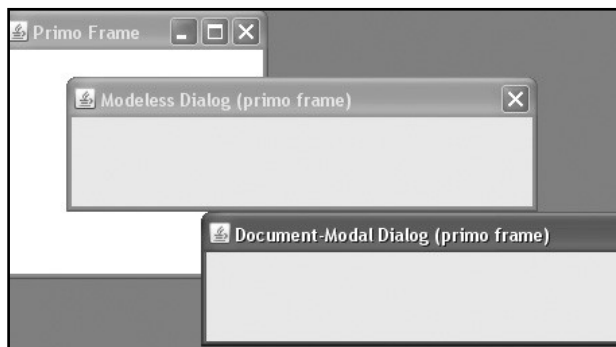



Figura 2.8: Due frame e due dialog box, di cui una di tipo Document/Modal

```
JDialog(Window owner, String title, Dialog.ModalityType modalityType)
```

Come per le altre nuove caratteristiche introdotte da Java 6, non è detto che la piattaforma grafica ospitante le supporti. Per saperlo è possibile ricorrere al metodo:

```
isModalityTypeSupported(modalityType)
```

Nel caso che una modalità non sia supportata, il fatto che venga usata ha lo stesso effetto del comportamento modeless.
Ecco, per esempio, una classe che istanzia due Frame: sul primo vengono create due dialog box, la prima Modeless, la seconda Document-Modal. A questo punto il programma aspetta che l'utente chiuda la finestra modale per proseguire (ma nel frattempo vengono "bloccate" solo le finestre del primo frame e della finestra modeless, che sono "parenti" della modale; il secondo frame può ricevere il focus o la selezione, **Figura 2.8**). Quando l'utente chiude la finestra modale viene aperta un'altra finestra modale: questa volta Application-Modal. Entrambi i frame risultano non selezionabili né possono ricevere il focus finché questa non viene chiusa.

```
package it.ioprogrammo.java6.desktop;
import java.awt.*;
import java.awt.event.*;
import sun.awt.*;

public class TestModality {

 private static WindowListener closeWindow = new WindowAdapter() {
   public void windowClosing(WindowEvent e) {
     e.getWindow().dispose();
   }
 };

 public static void main(String[] args) {
   Frame frame1 = new Frame("Primo Frame");
   frame1.setBounds(50, 50, 200, 200);
   frame1.addWindowListener(closeWindow);
   frame1.setVisible(true);
   Frame frame2 = new Frame("Secondo Frame");
   frame2.setBounds(600, 400, 200, 200);
   frame2.addWindowListener(closeWindow);
   frame2.setVisible( true );

   Dialog dialogModeless =
     new Dialog(
      frame1,
      "Modeless Dialog (primo frame)"
     );
   dialogModeless.setBounds(100, 100, 350, 100);
   dialogModeless.addWindowListener(closeWindow);
   dialogModeless.setVisible(true);

   Dialog dialogDocument =
```

```
    new Dialog((Window)dialogModeless,
     "Document-Modal Dialog (primo frame)",
     Dialog.ModalityType.DOCUMENT_MODAL
    );

  dialogDocument.setBounds(200, 200, 350, 100);
  dialogDocument.addWindowListener(closeWindow);
  dialogDocument.setVisible(true);

  Dialog dialogApplication =
    new Dialog(
      frame2,
      "Application-Modal Dialog (Secondo frame)",
      Dialog.ModalityType.APPLICATION_MODAL
    );
  dialogApplication.setBounds(500, 300, 350, 100);
  dialogApplication.addWindowListener(closeWindow);
  dialogApplication.setVisible(true);

 }
}
```


2.9 MODAL EXCLUSION

Per gestire le finestre modali è stata introdotta anche un'altra classe: java.awt.Dialog.ModalExclusionType.
In pratica è possibile indicare se una finestra è esclusa dal comportamento modale; per esempio, se si usa:

```
Frame form3 = new Frame("Frame escluso");
form3.setModalExclusionType(
  Dialog.ModalExclusionType.APPLICATION_EXCLUDE
);
```

Tale frame non viene bloccato da un'eventuale finestra modale a livello di applicazione. Anche per questa caratteristica è opportuno verificare che il gestore di finestre sottostante supporti questa caratteristica; per farlo:

```
isModalExclusionTypeSupported(modalExclusionType)
```

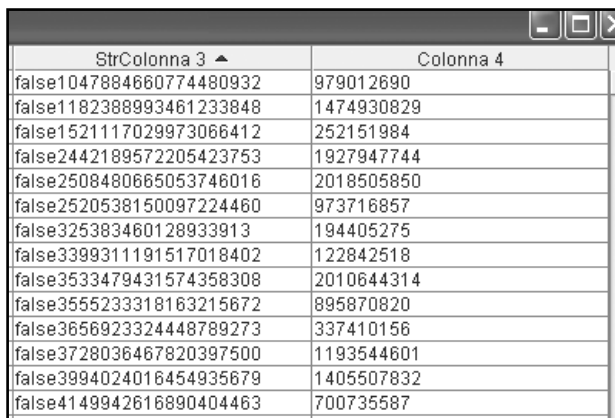
Ad ogni modo se la modalità di esclusione non è supportata, l'esclusione non ha effetto.

2.10 UNA JTABLE...ORDINATA!

In Java 6 è possibile applicare un ordine agli elementi visualizzati in una JTable; in particolare è possibile creare un oggetto di tipo TableRowSorter: passandogli un opportuno TableModel; ecco un programma che ne mostra l'uso:

```
public class TestJTable {

 private static WindowListener closeWindow = new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
       e.getWindow().dispose();
    }
 };

 public static void main(String args[]) {
  JFrame frame = new JFrame("Un esempio di JTable con ordinamento");
  frame.addWindowListener( closeWindow );
  String intestazione[] = {
    "StrColonna 1", "Colonna 2",
    "StrColonna 3", "Colonna 4"
  };
  Object righe[][] = TestJTable.getRandomData( intestazione );
```

```
TableModel model =
  new DefaultTableModel(righe, intestazione);

JTable table = new JTable(model);
TableRowSorter<TableModel> sorter =
  new TableRowSorter<TableModel>(model);
table.setRowSorter(sorter);
JScrollPane pane = new JScrollPane(table);
frame.add(pane);
frame.setSize(800, 600);
frame.setVisible(true);
 }
}
```

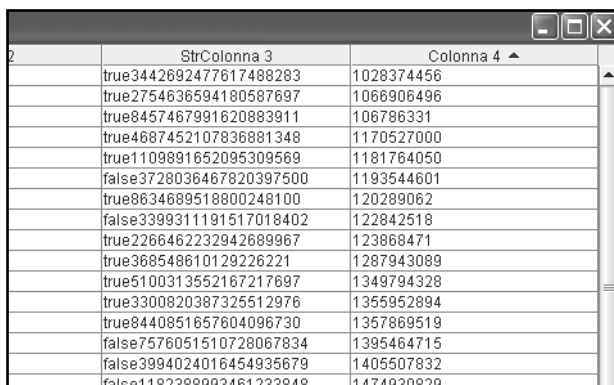
Facendo clic su un'intestazione, gli elementi vengono ordinati. In **Figura 2.9** si può notare l'ordinamento per la terza colonna. Facendo un secondo clic si può invertire l'ordine.

StrColonna 3 ▲	Colonna 4
false1047884660774480932	979012690
false1182388993461233848	1474930829
false1521117029973066412	252151984
false2442189572205423753	1927947744
false2508480665053746016	2018505850
false2520538150097224460	973716857
false325383460128933913	194405275
false3399311191517018402	122842518
false3533479431574358308	2010644314
false3555233318163215672	895870820
false3656923324448789273	337410156
false3728036467820397500	1193544601
false3994024016454935679	1405507832
false4149942616890404463	700735587

Figura 2.9: Ordine della visualizzazione degli elementi per la terza colonna

I problemi nascono per i tipi di dato numerici. Infatti, creando un siffatto TableRowSorter, di default viene sempre applicato l'ordinamento lessicografico (Figura 2.10); per i numeri questo tipo di ordinamento non è adatto. Per far sì che ogni colonna abbia un ordinamento adatto al suo tipo di dato si potrebbe applicare il seguente modello:

```
TableModel model =
 new DefaultTableModel(righe, intestazione){
   public Class getColumnClass(int col) {
    Class ris;
    if ((col>= 0) && (col<getColumnCount())) {
      ris = getValueAt(0, col).getClass();
    } else {
      ris = Object.class;
    }
    return ris;
   }
};
```

2	StrColonna 3	Colonna 4 ▲
	true3442692477617488283	1028374456
	true2754636594180587697	1066906496
	true8457467991620883911	106786331
	true4687452107836881348	1170527000
	true1109891652095309569	1181764050
	false3728036467820397500	1193544601
	true8634689518800248100	120289062
	false3399311191517018402	122842518
	true2266462232942689967	123868471
	true3685486101292226221	1287943089
	true5100313552167217697	1349794328
	true3300820387325512976	1355952894
	true8440851657604096730	1357869519
	false7576051510728067834	1395464715
	false3994024016454935679	1405507832
	false1182388993461233848	1474930829

Figura 2.10: L'ordinamento di default non è appropriato per i valori numerici

Ecco, infine, la routine usata per la generazione dei dati random, utile solo per mostrare un certo numero di dati per riempire la tabella (genera tra le 50 e le 250 righe; i valori delle la cui intestazione inizia per "Str" sono di tipo String, per le altre colonne usa dei valori interi positivi):

```
public static Object[][]
   getRandomData(String[] intestazione){
  java.util.Random rnd = new java.util.Random();
  int righe = Math.abs( rnd.nextInt() % 200 + 50 );
  Object[][] ris = new Object[righe][intestazione.length];
  for(int riga=0; riga<righe; riga++)
    ris[riga] = getRandomStr(intestazione);
  return ris;
}

private static Object[] getRandomStr(String[] intestazione){
  java.util.Random rnd = new java.util.Random();
  Object[] ris = new Object[intestazione.length];
  for(int elemento=0; elemento<intestazione.length; elemento++)
    if (intestazione[elemento].startsWith("Str"))
      ris[elemento] = rnd.nextBoolean()+" "+
        Math.abs( rnd.nextLong() );
    else
      ris[elemento] = Math.abs( rnd.nextInt() );
    return ris;
  }
```

2.11 JTABLE CON FILTRI

Una JTable, oltre a poter visualizzare i dati in maniera ordinata, li può anche filtrare per mostrare solo certi elementi che soddisfano a determinate condizioni. Queste condizioni sono espresse attraverso

oggetti di tipo RowFilter; tali oggetti possono essere costruiti usando uno dei seguenti metodi:

```
RowFilter.dateFilter( tipoConfronto, valoreRiferimento, colonna)
RowFilter.numberFilter( tipoConfronto, valoreRiferimento, colonna)
RowFilter.regexFilter( espressioneRegolare, colonna)
```

Il primo si applica a date, il secondo a numeri e il terzo a stringhe e permette di specificare una qualsiasi espressione regolare (per maggiori informazioni sulle espressioni regolari si può far riferimento alla pagina http://java.sun.com/docs/books/tutorial/essential/regex/). Per i primi due il tipo di confronto può essere una delle seguenti costanti:

```
RowFilter.ComparisonType.AFTER
RowFilter.ComparisonType.BEFORE
RowFilter.ComparisonType.EQUAL
RowFilter.ComparisonType.NOT_EQUAL
```

Il numero di colonna, comune a tutti i metodi, indica a quale colonna della tabella il filtro specificato vada applicato. È possibile anche costruire un numero qualsiasi di filtri e comporli usando uno dei seguenti metodi di composizione logica:

```
RowFilter.andFilter( filtri );
RowFilter.orFilter( filtri );
```

Il primo mette in AND tutti i filtri passati come argomento (in pratica verranno visualizzate solo le righe che li soddisfano tutti); il secondo li mette in OR (basterà che venga soddisfatto un solo dei filtri affinché la riga compaia nei risultati visualizzati). È anche possibile applicare un filtro "negativo" rispetto ad uno già creato usando il metodo (equivalente al metodo NOT logico):

```
RowFilter.notFilter(filtro);
```

Un oggetto RowFilter può essere applicato ad un oggetto TableRowSorter attraverso il metodo setRowFilter. Ecco, per esempio, come estendere l'esempio precedente affinché mostri solo le righe per cui la prima riga inizia con "true", la terza con "false" e la seconda è maggiore di 1.000.000.000 (**Figura 2.11**):

```
java.util.List<RowFilter<Object, Object>> filtri =
  new java.util.ArrayList<RowFilter<Object, Object>>();
filtri.add( RowFilter.regexFilter( "^true", 0) );
filtri.add(
  RowFilter.numberFilter(
    RowFilter.ComparisonType.AFTER,
    new Integer(1000000000),
    1)
  );
filtri.add( RowFilter.regexFilter( "^false", 2) );
sorter.setRowFilter( RowFilter.andFilter( filtri ) );
```

Un esempio di JTable con ordinamento

StrColonna 1	Colonna 2 ▲	StrC
rue4466468276097146987	1183384406	false46735987
rue2983219004149160374	1191238286	false21885814
rue3782726445178775436	1238571384	false36724510
rue5895388005879150367	1268108362	false63708063
rue3152834801256090216	1272755028	false29136648
rue6253374201537370869	1289293241	false29311666
rue6335461602060399460	1353557491	false47632336
rue3021166246354074530	1387696241	false39347417
rue5683748608759533335	1405498708	false89853146
rue7432063417032083672	1426028363	false54403513
rue3855824120265662638	1473377570	false20960100
rue3317018930165950035	1527458933	false38179909
rue5296621453805209384	1623822349	false40850893
rue3244720358932096648	1688208637	false91046564
rue9100864306299246229	1710575777	false73791329
rue2377808769903150394	1763766452	false35329097

Figura 2.11: Vengono visualizzati solo i dati che soddisfano i filtri impostati

2.12 SWING E I THREAD

Swing possiede una caratteristica che, se ignorata, può portare a scrivere del codice instabile: il disegno dei componenti a video (compreso il refresh degli stati dei componenti, come selezioni, input, e così via) viene gestito da uno specifico thread, chiamato "Swing event dispatch thread" (d'ora in poi EDT) che, e questa è la fonte dei problemi, coincide con il thread principale con cui viene eseguito il programma (infatti, di solito, il programma ha un main il quale, molto spesso, ha come unico compito quello di settare e istanziare la GUI e poi termina; quindi il thread EDT è, a questo punto, quello principale),

Ora ci si immagini la situazione in cui si effettua una connessione remota per reperire dei dati (per esempio verso un server che, a sua volta, ospita una servlet che interrogerà una base di dati remota). Se non vengono adottati particolari accorgimenti, mentre si è in attesa del risultato l'intera interfaccia grafica risulterà "congelata". In questa situazione l'utente può chiedersi il motivo di questo blocco pensando che il programma stia andando in crash. Ecco che inizia a fare clic per chiudere le finestre attive, premere tasti in cerca di un modo per bloccare l'operazione che sembra (sembra!) essere erroneamente bloccata. L'errore, in questo caso, è del programmatore che non ha previsto, per le operazioni che non restituiscono risultati nel brevissimo tempo, un thread separato.

2.13 SWINGWORKER

Java 6 introduce finalmente un costrutto che permette in maniera molto semplice l'esecuzione di un'operazione su un thread separato da quello principlae: SwingWorker.

Per un'analisi dettagliata sul suo utilizzo si può far riferimento all'articolo "Improve Application Performance With SwingWorker in Java SE 6", reperibile alla pagina http://java.sun.com/developer/technicalArticles/javase/swingworker/. In esso viene mostrata un'in-

tera applicazione che fa uso di Web Services e che realizza le invocazioni per il reperimento di immagini (e quindi molto "pesanti" in termini di byte da scaricare!) e aggiorna l'interfaccia grafica in maniera "asincrona" grazie a classi che estendono SwingWorker, lasciando all'utente libertà d'uso dell'applicazione. In effetti i Web Services sono, al giorno d'oggi, presenti in molte realtà enterprise. Anche in questo campo Java 6 possiede tante e interessanti novità…

XML & WEB SERVICES

Java 6 possiede già, al suo interno, il supporto nativo ai Web Services. Questo supporto non è, come si vedrà, completo, ma è sufficientemente semplice da usare anche a chi è profano della problematica e permette di creare in maniera molto semplice dei Web Services funzionanti, senza dover per forza installare componenti aggiuntivi né tantomeno conoscere le tecnologie sottostanti (anche se quest'ultimo tipo di conoscenza aiuta sia ad usarli in maniera consapevole che a risolvere eventuali problemi di interoperabilità). I Web Services si basano su numerose tecnologie standard, prima fra tutte l'XML. Nel capitolo si vedranno anche le tante (e notevoli!) novità di Java 6 per la gestione e la trasformazione di documenti XML e interessanti aggiunte per la gestione di firme digitali.

3.1 L'ESSENZA DEI WEB SERVICES

La creazione di Web Services consta in due parti: una server e una client. La server è la parte che realizza una determinata funzionalità, la client è quella che ne fa uso. Parlando genericamente di Web Services di solito si indica la parte server (per riferirsi alla parte client si usa riferirsi a "Web Service client"). Client e server possono stare su macchine diver-

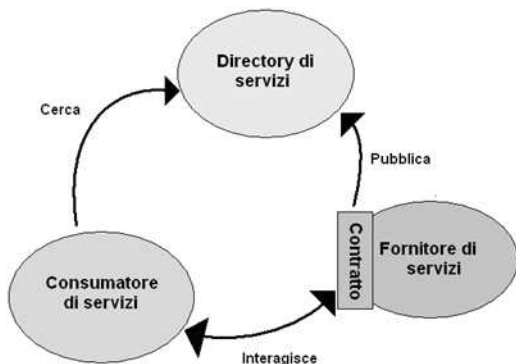



Figura 3.1: Un possibile scenario d'uso di Web Services

se, localizzate su sistemi diversi e, quel che più conta, possono essere scritte con tecnologie e piattaforme software completamente diverse. Infatti l'interscambio (invocazioni, sincrone o asincrone, e i risultati) dei dati avviene usando tecnologie indipendenti da uno specifico linguaggio di programmazione. Negli anni questo semplice scenario è evoluto in maniera che chi richiede un servizio (client) interroga un server il quale fornisce un risultato secondo formati e specifiche Standard (**Figura 3.1**). Da questo schema di base è possibile che il fornitore, a sua volta, faccia da client per uno o più servizi che concorrono a fornire la sua risposta; in questo modo esistono più attori, ciascuno specializzato in un compito, ma tutti contribuiscono a soddisfare l'invocazione originaria (si veda la **Figura 3.2** per uno schema di massima di questa architettua che si chiama SOA, Service Oriented Archiecture).

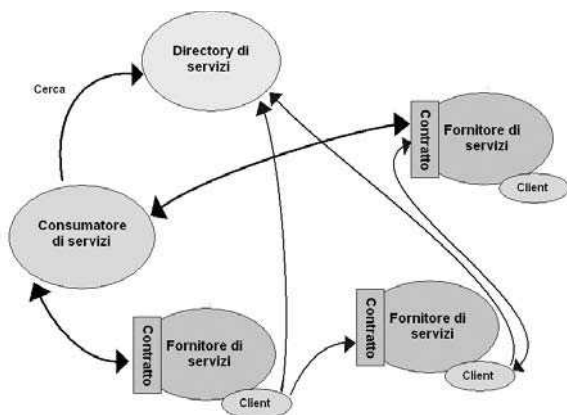



Figura 3.2: Più fornitori in un’architettura a servizi distribuita (SOA)

Questo tipo di architettura, del tutto multipiattaforma e interoperabile, ha avuto un enorme successo: al giorno d'oggi ogni linguaggio di programmazione possiede uno o più framework che permettono la realizzazione di Web Services (sia server che client). Non solo: tutti i principa-

li fornitori di servizi (come eBay, Amazon e lo stesso Google) offrono Web Services per interagire con i loro servizi (e gestire, per esempio, aste online, ordinare libri ed effettuare le usuali ricerche nelle proprie applicazioni!). Il vantaggio per i programmatori è evidente: ci si appoggia ad un servizio esterno per svolgere delle attività specifiche. Ma è vantaggioso anche per chi offre il servizio: deve definire e implementare un unico Web Service, molto generale, e grazie ad esso i servizi sono subito disponibili verso innumerevoli applicazioni che possono essere scritte in un linguaggio a scelta dello sviluppatore senza vincoli né altre limitazioni.

3.2 QUALI TECNOLOGIE

Presentare tutte le tecnologie sottostanti i Web Services in pochissimo spazio non è certamente esaustivo, ma vale la pena accennare alle nozioni che è necessario possedere per capire cosa accade nella comunicazione tra un server e (uno o più) client secondo un'architettura a Web Services (maggiori dettagli in [Venuti, 2005]).

3.3 IL PROBLEMA DI BASE

Alla nascita i Web Services si son posti il problema dell'interoperabilità. Per questo è stato scartato ogni formato binario e proprietario; è stato scelto quello che era, da anni, considerato il formato di interscambio dei dati più diffuso: l'XML (anche se prima di allora tale formato era usato per lo più per scambiare dati provenienti da database eterogenei o come tecnologia per veicolare i dati verso client Web).
XML come lingua franca
Un documento XML è un semplice documento testuale in cui i valori sono racchiusi tra opportuni tag. L'insieme dei tag ammessi può essere definito da un apposito documento. In origine c'erano i DTD, mentre ad oggi il formato di descrizione più diffuso è l'XML Schema. Un documento XML contiene soli dati e riferimenti ad un eventuale XML Sche-

ma, ma non contiene alcuna informazione di "presentazione". L'XML Schema definisce sia i tag ammessi che i valori possibili all'interno di ogni tag (con eventuali regole sintattiche e semantiche che essi devono rispettare). Si parla di XML ben formato quando rispetta le regole sintattiche XML; si dice valido quando, dato uno schema di validazione, il documento è coerente con le sue regole (in questo senso un documento XML è ben formato o meno in assoluto, è valido rispetto ad uno o più schema o DTD).

3.4 SOAP

L'XML da solo non basta: è stato definito un "involucro" secondo un nuovo standard, chiamato SOAP (Simple Object Access Protocol, la cui pagina di riferimento della versione 1.1 è http://www.w3.org/TR/2000/NOTE-SOAP-20000508/; per la versione 1.2 si faccia riferimento alle pagine http://www.w3.org/TR/soap12-part1/ e http://www.w3.org/TR/soap12-part2/). Tale involucro si presenta in due parti: uno header e un corpo (Figura 3.3). L'header contiene informazioni di trasporto, autenticazione o quant'altro utile a localizzare sia i destinatari che eventuali azio-

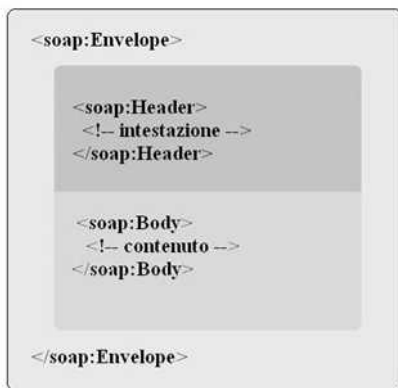



Figura 3.3: Un messaggio SOAP nelle due componenti fondamentali

ni da intraprendere per leggere il corpo. Nel corpo ci stanno i dati veri e propri. Ad ogni modo ogni sua parte deve essere XML standard.

3.5 UN CONTRATTO? WSDL!

Conoscere che esiste un qualche servizio SOAP non è sufficiente. Si vorrebbe sapere, in dettaglio, sia il formato dei messaggi ammessi (ovvero validi), sia i protocolli di comunicazione accettati che eventuali URI d'accesso. Tutto questo dovrebbe essere descritto in un formato standard affinché diverse piattaforme software possano farvi riferimento: nasce così il formato WSDL (Web Services Definition Language 1.1, pagina di riferimento http://www.w3.org/TR/wsdl). Esso dovrebbe essere così dettagliato da permettere la creazione automatica di classi da parte di un framework che, opportunamente istanziate, possono instaurare una connessione al servizio senza interventi di configurazione "manuale". In questo senso un WSDL deve descrivere tutti i metodi esposti, i parametri di ingresso e uscita e ogni altra informazione utile. Anche un documento WSDL è suddiviso in sezioni logiche che si possono così sintetizzare (**Figura 3.4**):

- **Types:** è un contenitore di tipi di dato definiti attraverso un opportuno meccanismo di specifica (di solito XSD);
- **Message:** una definizione astratta dei dati che sono scambiati nella forma di messaggi; la definizione comprende la specifica dei tipi coinvolti nella comunicazione;
- **Operation:** le azioni esposte dal servizio (definizione astratta);
- **Port Type:** un insieme di operazioni supportate da uno o più end-

Figura 3.4: Le componenti fondamentali di un documento WSDL

point (insieme astratto);

- **Binding:** un protocollo e una specifica sui formati dei dati concreti per un particolare Port Type;
- **Port:** un endpoint (singolo) definito come combinazione di binding e indirizzi di rete;

Attenzione
Con il termine endpoint si è soliti riferirsi ad una URI a cui risponde il servizio. In parole povere, nel caso di http, è l'indirizzo da interrogare per ottenere i risultati dal servizio, passandogli opportuni dati di ingresso.

- **Service:** una collezione di endpoint correlati.

3.6 UDDI: LE DIRECTORY PER I SERVIZI

Avendo a disposizione un WSDL in teoria permette la piena fruizione di un servizio. Però potrebbe nascere un'ulteriore esigenza: quella di ricercare un servizio partendo da esigenze espresse in maniera formale (per esempio scegliere un servizio di autenticazione tra tanti disponibili). Nasce l'idea, mutuata dalle directory di siti Web, di usare un qualche "accentratore" che permetta il censimento di un servizio e il suo reperimento in seguito a opportune interrogazioni. È per soddisfare questa esigenza che nasce lo standard UDDI.

Attenzione
In moltissimi servizi Web sviluppati ad hoc per clienti e istituzioni, non c'è la necessità di ricorrere al loro censimento su un server UDDI. Per questo motivo di solito è sufficiente fornire un WSDL del servizio per permettere la creazione di opportuni client per il suo utilizzo.

3.7 IL TRASPORTO

Le tecnologie sopracitate non si riferiscono ad alcun meccanismo di trasporto specifico. Infatti i Web Services sono indipendenti da esso; eppure il meccanismo di trasporto più usato è http o https. Gli esempi che illustreremo faranno uso di questo standard, ma va tenuto presente che si potrebbero usare anche altri meccanismi di trasporto quali FTP, SMTP e così via.

3.8 JAVA E IL SUPPORTO AI WS

Java ha avuto, storicamente, diversi packages per la gestione dei Web Services; ogni package ha una sua funzione specifica. Però questi packages sono sempre stati esterni al JDK (quindi non presenti nelle diverse distribuzioni) e andavano installati a parte (il pacchetto che si doveva installare era il JWSDP: Java Web Services Developer Pack). Java 6, per la prima volta, offre molte di queste librerie, prima opzionali, all'interno del JDK stesso! Ecco come usarle, sia con i tool del JDK che usando un IDE evoluto come NetBeans 5.5.

Package	Tecnologia (JSR)	Utilizzo
javax.xml.ws	JAX-WS (224)	La nuova versione dello standard JAX-RPC per lo sviluppo dei Web Services (la prima versione di JAX-WS disponibile è la 2.0)
javax.jws	WebServices Metadata (181)	Introduce metadati e annotazioni per esprimerli per semplificare lo sviluppo di Web Services
javax.xml.soap	SAAJ (67)	SOAP with Attachments API for Java: permette di inviare documenti XML in maniera standard dalla piattaforma Java

Tabella 3.1 I nuovi package, specifici per i Web Services, introdotti in Java 6.

3.9 CREARE WEB SERVICE IN JAVA 6

Le API a disposizione di Java 6 per la creazione dei Web Services fanno parte dei package descritti in **Tabella 3.1** (accanto alle tecnologie/spe-

Attenzione

Il nome JAX-WS è stato scelto al posto di JAX-RPC (Java API for XML Based RPC). Infatti JAX-RPC 2.0 è stato chiamato JAX-WS 2.0 in quanto l'uso di RPC nel nome poteva far credere che la specifica si basasse esclusivamente su RPC, cosa non vera. Inoltre sono state rimarcate, con il cambio di nome, alcune incompatibilità con il passato (per esempio JAXB ora è usato per tutti i binding). L'uso di package diversi permette di far coesistere servizi diversi, alcuni sviluppati con JAX-RPC, altri con la nuova JAX-WS. Per i dettagli si veda http://weblogs.java.net/blog/kohlert/archive/2005/05/jaxrpc_20_renam.html.

cifiche di riferimento). Si introduce ora l'uso di WebServices Medata e JAX-WS realizzando un semplice Web Service. Dapprima fornendo la parte server, successivamente creando un opportuno client.

3.10 QUALI FUNZIONALITÀ?

La primissima cosa da definire sono le funzionalità esposte dal nuovo Web Service. Per farlo si può iniziare con il creare una classe che offre i metodi che si vuol rendere accessibili da remoto attraverso Web Services. Si crei, per esempio, una classe che offre due metodi: il primo restituisce true se la stringa passata è un codice fiscale valido, il secondo restituisce true se la stringa è una partita IVA valida:

```
package it.ioprogrammo.java6.ws;
public class primoWS{
  public boolean isCodiceFiscaleValido(String codiceFiscale){
    // algoritmo di verifica
  }
```

```
public boolean isPartitaIvaValida(String partitaIva){
  // algoritmo di verifica
 }
}
```

Questa classe potrebbe far parte di un framework, magari distribuito come file JAR, e potrebbe essere utilizzata in diversi programmi Java. Renderlo un Web Service può portare a dei vantaggi immediati come, per esempio, "vendere" il servizio a chiunque necessiti un simile controllo; non solo: rendendolo disponibile come Web Services e non come componente, si permette che esso venga usato anche da client di lingauggi diversi (come può essere un programma scritto in VB.NET o Perl!).

3.11 REALIZZARE LA PARTE SERVER

La bellezza di JAX-WS è che un nuovo Web Service che si basa su una classe esistente può essere scritto usando la classe e aggiungendovi solo delle opportune annotazioni; per la classe precedente basterà inserire @javax.jws .WebService prima della definizione della classe e… si è finito!

```
@javax.jws.WebService
public class primoWS{
 // Come prima
}
```

Attenzione
È importante che la classe appartenga ad un package per usare la notazione @WebService senza ulteriori specifiche. Si vedrà in seguito il motivo di questo vincolo (essenzialmente è dovuto alla convenzione d'uso del nome del package per i namespace del WSDL risultante).

3.12 COMPILARE IL WEB SERVICE

La classe va compilata come sempre usando il compilare del JDK:

```
>javac it\ioprogrammo\java6\ws\primoWS.java
```

ma, sul file .class risultante, va applicato il comando wsgen (presente nella cartella bin/ del JDK):

```
>wsgen -classpath . it.ioprogrammo.java6.ws.primoWS
```

Il risultato di quest'ultimo comando è la creazione di una cartella, javaws, dove si trova il file .class. Tale nuova cartella contiene una coppia di file per ciascun metodo esposto nel Web Service: un file si chiama proprio come il metodo, l'altro come il metodo seguito da "Response". Ciascun file così ottenuto è presente sia in forma sorgente (.java) che compilata (.class). In **Figura 3.5** gli otto file risultanti per l'esempio appena scritto. In **Tabella 3.2** tutte le opzioni che è possibile specificare usando il comando wsgen per personalizzarne il risultato.
Un modo per semplificare i due comandi (javac e wsgen) è quello di usare apt:

```
> apt -classpath . it\ioprogrammo\java6\ws\primoWS.java
```

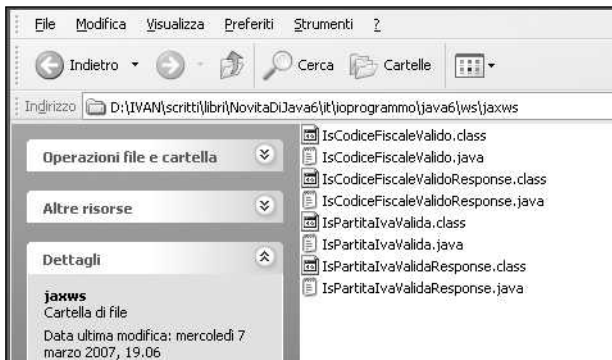



Figura 3.5: I file generatidal comando wsgen per il servizio d'esempio

Opzione	Significato
-classpath <path>	Specifica dove trovare i file .class
-cp <path>	Un modo più compatto ma identico a -classpath
-d <cartella>	Indica dove posizionare i file (compilati) generati
-extension	Eventuali opzioni non standard e specifiche per questa implementazione del tool
-help	Mostra l'aiuto in linea
-keep	Se usata, fa sì che vengano mantenuti i file sorgente (.java) delle classi generate
-r <cartella>	Da usarsi insieme all'opzione -wsdl e specifica dove posizionare i file di risorse generati (come i file WSDL)
-s <cartella>	Indica dove posizionare i file sorgente generati (se omessa usa la stessa cartella specificata con -d)
-verbose	Stampa i messaggi del compilatore
-version	Stampa la versione del tool
-wsdl[:protocollo]	Normalmente wsgen non genera file WSDL, a meno che non venga specificata questa opzione. Il protocollo è opzionale e, se specificato, viene usato per valorizzare wsdl:binding (esempi di protocollo validi sono soap1.1, quello usato di default, e Xsoap1.2)
-servicename <nome>	Da usarsi insieme a -wsdl e specifica il valore che assume wsdl:service
-portname <nome>	Da usarsi insieme a -wsdl e specifica il valore da assegnare a wsdl:port

Tabella 3.2 Le opzioni del comando wsgen.

In **Tabella 3.3** alcune tra le opzioni che è possibile specificare usando il comando apt. Il risultato del precedente comando è identico all'applicazione combinata di javac e wsgen.

Attenzione
apt non è considerato un comando standard in quanto nelle future versioni si potrebbe fare a meno di esso. Per questo motivo va usato con cautela in eventuali ambienti di sviluppo che si vorrà far evolvere utilizzando le nuove versioni di Java.

Opzione	Significato
-classpath <path>	Specifica dove reperire i file .class (si può usarlo anche nella forma compatta –cp)
-d <path>	Indica dove salvare i file compilati generati
-g	Genera tutte le informazioni di debug
-g:none	Sopprime tutte le informazioni di debug
-help	Stampa l'help completo
nocompile	Indica di non compilare i file sorgente generati
-s <path>	Indica dove salvare i file sorgente generati
-source <versione>	Indica di mantenere file java compatibili con la versione specificata
-sourcepath <path>	Indica dove recuperare i file sorgente
-version	Stampa le informazioni sulla versione del tool

Tabella 3.3 Alcune delle opzioni del comando apt.

Osservando i file generati, si può notare come essi facciano uso del package javax.xml.bind.annotation sia per la specifica dei parametri dei metodi:

```
package it.ioprogrammo.java6.ws.jaxws;

import javax.xml.bind.annotation.*;

@XmlRootElement(name = "isCodiceFiscaleValido", namespace
                                        = "http://ws.java6.ioprogrammo.it/")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "isCodiceFiscaleValido", namespace
                                        = "http://ws.java6.ioprogrammo.it/")
public class IsCodiceFiscaleValido {
    @XmlElement(name = "arg0", namespace = "")
    private String arg0;

    public String getArg0() {
```

```
        return this.arg0;
    }

    public void setArg0(String arg0) {
        this.arg0 = arg0;
    }
}
```

Che per il loro risultato (il caso delle classi che si chiamano come il metodo ma con il suffisso “Response”):

```
package it.ioprogrammo.java6.ws.jaxws;

import javax.xml.bind.annotation.*;

@XmlRootElement(name = "isCodiceFiscaleValidoResponse", namespace
                                        = "http://ws.java6.ioprogrammo.it/")
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "isCodiceFiscaleValidoResponse", namespace =
                                        "http://ws.java6.ioprogrammo.it/")
public class IsCodiceFiscaleValidoResponse {

    @XmlElement(name = "return", namespace = "")
    private boolean _return;

    public boolean get_return() {
        return this._return;
    }

    public void set_return(boolean _return) {
        this._return = _return;
    }
}
```

Si può notare che il namespace è ricavato dal nome del package a cui appartiene la classe di partenza. Nel seguito si vedrà come usare altre annotazioni per personalizzare questo e altri aspetti del Web Service generato.

Attenzione
L'uso dei comandi wsgen e apt può essere reso trasparente da un IDE che supporta le nuove annotazioni. NetBeans 5.5, per esempio, esegue questi comandi "dietro le quinte" e permette allo sviluppatore la sola specifica delle annotazioni. L'IDE si occupa di tutto il resto.

3.13 PUBBLICARE IL SERVIZIO

Ora che si è realizzato un Web Service, esso va "esposto" affinché possa venire invocato. In pratica un servizio deve poter rispondere a delle invocazioni dei client attraverso un determinato protocollo. È interessante osservare che il protocollo può essere un qualsiasi protocollo di comunicazione: http (o https), SMTP e così via. Spesso i protocolli di riferimento sono http/https. Per pubblicare un servizio con tale protocollo si può installare un qualunque Web Server che accetti programmi Java (per esempio Tomcat) oppure si fa uso di un server che, proprio a partire da Java 6, è integrato nel JDK. Per far uso del server integrato e pubblicare il servizio precedente si può ricorrere alla classe javax.xml.ws.Endpoint (anche questa è una nuova classe introdotta in Java 6!). Il metodo da usare è publish con due argomenti: il primo è l'url a cui risponderà il servizio (è importante che la parte finale dell'url coincida con il nome della classe!) e poi l'instanza creata della classe che espone il servizio:

```
package it.ioprogrammo.java6.ws;

public class eseguiWS{
```

```
  public static final void main(String[] args){
        javax.xml.ws.Endpoint.publish(
                "http://localhost:8080/miows/primoWS", new primoWS()
        );
  }
}
```

Per eseguirla:

```
>java -classpath .;%CLASSPATH% it.ioprogrammo.java6.ws.eseguiWS
```

Però se si prova ad accedere all'url http://localhost:8080/miows/primoWS non si ottiene alcun risultato (o meglio: non viene visualizzato alcunché). Il motivo è che un servizio Web deve essere invocato da un client che gli fornisce un opportuno messaggio SOAP; la richiesta http del browser non è appropriata. Come fare a verificare che il servizio è effettivamente pubblicato? Si può ricorrere ad un altro tipo di interrogazione: la richiesta dei dettagli ovvero del documento WSDL associato al servizio. Tale richiesta va fatta usando ?wsdl all'url precedente. Il risultato? Lo si vede in **Figura 3.6**!

Attenzione
Il programma eseguiWS resta attivo finché non si decide di terminare la sua esecuzione o distruggendo il processo o, nel caso sia stato eseguito da una console dei comandi, premendo la combinazione di tasti [CTRL]+[C].

3.14 CREARE UN CLIENT

Come detto un servizio Web va invocato con un opportuno client. Ora si realizzerà un client usando, come prima, le nuove caratteristiche di Java 6 per far meno lavoro possibile! Un client è un programma in grado di colloquiare con un Web Service pubblicato. Per creare in automatico

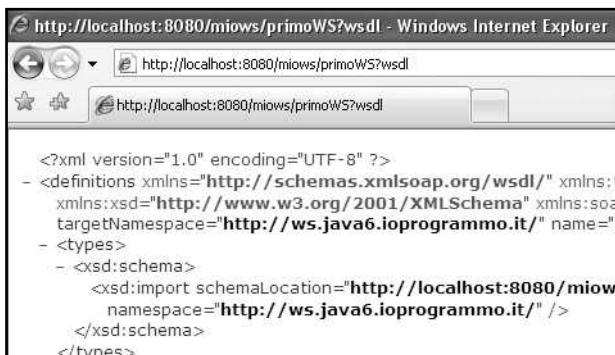



Figura 3.6: Il WSDL del servizio pubblicato: il server c'è

un client è fondamentale essere a conoscenza del documento WSDL che descrive il server; avendo il server attivo, si può ricorrere al tool wsimport in questo modo:

```
>wsimport -p mio.nome.client -d .\cartella_destinazione -keep http://
                                    localhost:8080/miows/primoWS?wsdl
```

Il parametro -p permette di specificare un proprio package, mentre -d la cartella di destinazione dove generare le classi. Il parametro -keep indica di generare anche i file sorgente (.java) e non solo quelli compilati (.class). In **Tabella 3.4** tutte le opzioni che è possibile specificare usando il comando wsimport. Se il WSDL non è disponibile su una pagina accessibile, ma è presente in locale (per esempio perché è stato precedentemente salvato su file system), esso va specificato come URI:

```
>wsimport -p mio.nome.client -d .\cartella_destinazione -keep qualeFile.wsdl
```

In **Figura 3.7** i diversi file generati per il client creato appositamente per il primo Web Service di prova. Ecco, a titolo di esempio, il contenuto del file mio.nome.client.primoWS.java (sono stati omessi i commenti generati dal tool):

Opzione	Significato
-d <nome cartella>	Specifica dove mettere i file, compilati, creati (attenzione: la cartella deve esistere!)
-b <path>	Specifica file specifici per il binding (file che possono essere validi secondo le specifiche JAX-WS o JAXB)
-B <opzione per jaxb>	Permette di passare l'opzione al compilatore JAXB schema
-catalog	Specifica file catalog (servono a risolvere referenze a entità esterne)
-extension	Eventuali parametri specifici dell'implementazione (e per questo non standard)
-help	Mostra l'aiuto in linea delle diverse opzioni
-httpproxy: <host>:<porta>	Usa uno specifico proxy server HTTP (se la porta non viene specificata, di defaul usa la 8080, quella standard per Tomcat)
-keep	Se c'è questa opzione vengono mantenuti anche i file sorgente (.java); senza questi vengono cancellati e restano solo i file compilati (.class)
-p	Permette di specificare un nome di package (se non viene usato le classi prendono il nome di package specificato o nel file WSDL oppure dal proprio schema personalizzato, se usato)
-s <directory>	Usare questa opzione insieme a keep per specificare una cartella diversa per i file sorgente (altrimenti viene usata la cartella specificata con l'opzione -d, dove vanno a finire i file .class)
-verbose	Stampa eventuali messaggi del compilatore
-version	Versione del tool
-wsdllocation <dove>	Dove reperire il file WSDL (può essere sia una URL che un file locale)
target	tilizza una specifica versione dello standard JAX-WS per i file creati
quiet	Nessun output del tool

Tabella 3.3 Le opzioni del comando wsimport.

```
package mio.nome.client;
import javax.jws.*;
```

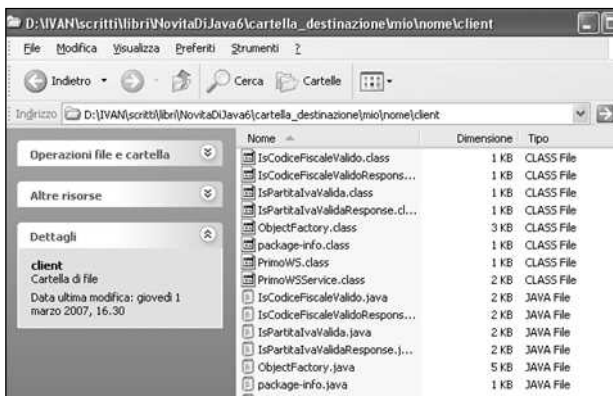



Figura 3.7: Tutti i file generati per la parte client del Web Services

```
import javax.xml.ws.*;

@WebService(name = "primoWS", targetNamespace =
                                         "http://ws.java6.ioprogrammo.it/")
public interface PrimoWS {

   @WebMethod
   @WebResult(targetNamespace = "")
   @RequestWrapper(localName = "isCodiceFiscaleValido",
    targetNamespace = "http://ws.java6.ioprogrammo.it/",
    className = "mio.nome.client.IsCodiceFiscaleValido")
   @ResponseWrapper(localName = "isCodiceFiscaleValidoResponse",
    targetNamespace = "http://ws.java6.ioprogrammo.it/",
    className = "mio.nome.client.IsCodiceFiscaleValidoResponse")

   public boolean isCodiceFiscaleValido(
      @WebParam(name = "arg0", targetNamespace = "")
      String arg0);

   @WebMethod
```

```
    @WebResult(targetNamespace = "")
    @RequestWrapper(localName = "isPartitaIvaValida",
      targetNamespace = "http://ws.java6.ioprogrammo.it/",
      className = "mio.nome.client.IsPartitaIvaValida")
    @ResponseWrapper(localName = "isPartitaIvaValidaResponse",
      targetNamespace = "http://ws.java6.ioprogrammo.it/",
      className = "mio.nome.client.IsPartitaIvaValidaResponse")
    public boolean isPartitaIvaValida(
        @WebParam(name = "arg0", targetNamespace = "")
        String arg0);
}
```

Si può notare che esistono molte più annotazioni di quelle illustrate in precedenza. Alcune di esse fanno parte della specifica JAX-WS che è la specifica di più basso livello usata per la creazione dei Web Services. Per fortuna usare il client è piuttosto semplice (e non serve certo guardare ai diversi file generati, anche se questo può aiutare a comprendere anche la tecnologia JAX-WS). Ecco com'è possibile invocare un metodo remoto, come può essere isCodiceFiscaleValido:

```
ws= (new mio.nome.client.PrimoWSService()).getPrimoWSPort();
ws.isCodiceFiscaleValido("CODICEFISCALE")
```

In questo caso l'URL a cui il client prova ad accedere non è specificata dal programma. In questo caso è la stessa di quella specificata nel file WSDL! Questa situazione di default potrebbe non andare sempre bene. Spesso i client, per esempio, vengono sviluppati in ambienti di test e poi portati in un ambiente di produzione: questo fa sì che l'URL a cui risponde il servizio remoto cambi nei due casi. Per fortuna è relativamente semplice anche passare una URL personalizzata. Ecco, per esempio, come far sì che si legga il primo parametro passato a linea di comando e lo si usa come nuova URL (nel caso non ci siano parametri viene usata l'URL di default, usando la stessa invocazione precedente):

```
package it.ioprogrammo.java6.ws;
public class classeTestClient {
  public static final void main(String[] args) throws Exception{
        mio.nome.client.PrimoWS ws = null;
        if (args.length==0)
            ws = (new mio.nome.client.PrimoWSService())
                 .getPrimoWSPort();
        else{
            ws = (mio.nome.client.PrimoWS)
              (new mio.nome.client.PrimoWSService(
                    new java.net.URL(args[0]),
                    new javax.xml.namespace.QName(
                        "http://ws.java6.ioprogrammo.it/",
                        "primoWSPort")
                    ).getPrimoWSPort());
        }
      System.out.println(
         ws.isCodiceFiscaleValido("qualeCodice")
      );
    }
}
```

L'URL a cui risponde un servizio Web è chiamato end-point. Si noti che per la costruzione di un end-point personalizzato è necessario creare sia un oggetto java.net.URL (a cui si può, per esempio, passare la stringa che identifica l'URL) e un oggetto javax.xml.namespace.QName; per quest'ultimo oggetto è necessario specificare sia il namespace da usare che il nome del servizio Web.

3.15 CREARE ANCHE IL SERVER DAL WSDL

Come fatto per la parte client è possibile ricavare anche il codice del

server usando wsimport. Di solito si utilizza questa strada in due situazioni: la prima è quando si decide di creare prima il WSDL che, in questo modo, può essere personalizzato e adattato "a mano", senza usare alcun tool automatico. Questa strada è piuttosto complessa e necessita di un'ottima padronanza di tutte le specifiche sottostanti i Web Services. La seconda situazione è quando il server non è disponibile e si vuole, in qualche modo, testare il client (si pensi al caso in cui un cliente possiede un server realizzato come Web Service e che ci dà, come unica specifica, il WSDL: prima di effettuare dei test presso il suo ambiente, che può essere di produzione, è preferibile fare dei test in locale predisponendo un server ad hoc).

3.16 VINCOLI

Ma tutte le classi possono diventare Web Services? Non proprio. Infatti devono essere rispettati questi (pochi!) vincoli:

- la classe deve essere pubblica;
- la classe deve essere concreta (quindi non astratta);
- la classe non deve essere dichiarata final;
- deve avere un costruttore pubblico senza parametri (nel caso non ci siano costruttori, si ricorda che Java ne crea uno di default proprio senza parametri);
- non deve definire un metodo chiamato finalize().

In fondo sono vincoli molto poco stringenti e non rappresentano certo una limitazione della tecnologia.

3.17 SOLO ALCUNI METODI

Come accennato in precedenza esistono ulteriori annotazioni per intervenire sulla personalizzazione dei diversi aspetti del Web Service. Si è visto che una qualsiasi classe Java (a meno dei vincoli appena citati!) può essere trasformata in Web Service usando la sola notazione @Web-

Service. Però c'è un dettaglio che potrebbe non andare bene: tutti i metodo pubblici (compresi eventuali metodi ereditati, a meno di quelli ereditati da Object) divengono servizi esposti all'esterno. Riprendendo in mano la classe primoWS (e copiandola in secondoWS) si vorrebbe esporre solo il primo metodo, isCodiceFiscaleValido, e non il secondo; per farlo basta introdurre la annotazione @WebMethod sul primo metodo:

```
package it.ioprogrammo.java6.ws;

@javax.jws.WebService
public class secondoWS{

 @javax.jws.WebMethod
 public boolean isCodiceFiscaleValido(String codiceFiscale){
    // algoritmo di verifica
 }
 public boolean isPartitaIvaValida(String partitaIva){
    // algoritmo di verifica
 }
}
```

In questo modo, in automatico, il secondo non risulta esposto all'esterno. Però ci sono ulteriori personalizzazioni, molto più puntuali e che permettono una personalizzazione molto avanzata…

3.18 UTILIZZO DEI METADATI

La annotazioni @WebService e @WebMethod possono far uso di appositi metadati che permettono una personalizzazione puntuale sia delle caratteristiche del servizio Web esposto che sui singoli metodi (e quindi hanno anche un impatto sul WSDL generato). Tutti i metadati utilizzabili in @WebService sono riportati in **Tabella 3.5**.
Ecco, per esempio, come usare un namespece diverso dal package e un proprio nome per il servizio:

Attenzione
Solo metodi pubblici possono essere annotati con @WebMethod; per i vincoli sui parametri del metodo, i valori restituiti e le possibili eccezioni si faccia riferimento alla sezione 3.6 della specifica JAX-WS.

```
@WebService(
  name = "MioServizioWeb",
  targetNamespace = "http://ivenuti.altervista.org/esempioWs"
)
```

Metadato	Significato
name	Specifica il nome del Web Service (quando non viene indicato il nome coincide con quello della classe). Nel caso si usi WSDL 1.1 è il nome assegnato a wsdl:portType
targetNamespace	Nome del namespace utilizzato per il servizio (di default prende il nome del package della classe)
serviceName	Il nome riferito al service del Web Service che, nel caso di WSDL 1.1, coincide con il nome assegnato a wsdl:service (se non specificato viene usato il nome della classe con il suffisso "Service"
portName	Con WSDL 1.1 è il valore specificato in wsdl:port
endpointInterface	Usato per separare l'interfaccia dalla classe che implementa il servizio. In pratica l'interfaccia (chiamata appunto endpointInterface) definisce solo i metodi, la classe la implementa. Far riferimento alla documentazione della specifica JSR 181 quando si usa questa opzione, in quanto ha un impatto anche sugli altri metadati qui elencati.
wsdlLocation	Indica qual è il WSDL di origine della classe (e a cui si vuol far riferimento senza far sì che l'implementazione generi il WSDL a partire dalla classe realizzata)

Tabella 3.5 I metadati per la personalizzazione della annotazione @javax.jws.WebService.

```
public class secondoWS {
  // ...
}
```

In maniera analoga anche la annotazione @WebMethod può utilizza-

Metadato	Significato
operationName	Specifica il nome dell'operazione (quando non viene indicato coincide con quello del metodo a cui si riferisce). Nel caso si usi WSDL 1.1 è il nome assegnato a wsdl:operation
action	Specifica la action del metodo che, nel caso di binding SOAP, equivale al soap action; se non specificata assume valore di stringa vuota
exclude	Se True, il metodo non verrà esposto all'esterno (di default è False). La domanda è: ma non basta non indicare @WebMethod per il metodo in questione? Per una classe che non estende altre questo è sufficiente; ma si pensi al caso in cui la superclasse definisca un metodo pubblico e con la annotazione @WebMethod; teoricamente tutte le sottoclassi non possono far altro che esporlo. Se una sottoclasse non lo vuol fare, usare questo metadato è l'unico modo per risolvere il problema. Se usato, non devono comparire altri metadati
portName	Con WSDL 1.1 è il valore specificato in wsdl:port
endpointInterface	Usato per separare l'interfaccia dalla classe che implementa il servizio. In pratica l'interfaccia (chiamata appunto endpointInterface) definisce solo i metodi, la classe la implementa. Far riferimento alla documentazione della specifica JSR 181 quando si usa questa opzione, in quanto ha un impatto anche sugli altri metadati qui elencati.
wsdlLocation	Indica qual è il WSDL di origine della classe (e a cui si vuol far riferimento senza far sì che l'implementazione generi il WSDL a partire dalla classe realizzata)

Tabella 3.6 I metadati per la personalizzazione della annotazione @javax.jws.WebMethod.

re appositi metadati per modificarne il comportamento di default. Essi sono sintetizzati in **Tabella 3.6**.

Ecco un esempio di personalizzazione della annotazione @WebMethod:

```
@WebService(
  name = "MioServizioWeb",
  targetNamespace = "http://ivenuti.altervista.org/esempioWs"
)
public class secondoWS {
   @WebMethod(operationName = "checkCodiceFiscale")
   public boolena isCodiceFiscaleValido(String input) {
     // ...
  }
}
```

Anche i singoli parametri di un metodo esposto come servizio Web possono possedere dei propri metadati; essi sono specificati grazie alla annotazione @WebParam. In questo caso quelli utilizzabili sono riportati in **Tabella 3.7**. Infine è possibile personalizzare con dei metadati il tipo restituito da un metodo che è stato reso pubblico come Web Servi-

Metadato	Significato
name	Nome da assegnare al parametro
partName	Il nome da dare a wsdl:part. DI default coincide con il metadato name
targetNamespace	XML Namespace per il parametro
mode	Tipo di parametro: IN è in solo ingresso, OUT in sola uscita, INOUT ingresso e uscita
header	Se vale True, il parametro fa parte dello header e non del body del messaggio (di default vale false)

Tabella 3.7 I metadati per la personalizzazione della annotazione @javax.jws.WebParam.

ce. In questo caso si usa la annotazione @WebResult, con uno dei metadati indicati in **Tabella 3.8**.

Metadato	Significato
name	Nome da assegnare al valore di ritorno
partName	Il nome da dare a wsdl:part. Di default coincide con il metadato name
targetNamespace	XML Namespace per il valore di ritorno
header	Se vale True, il parametro fa parte dello header e non del body del messaggio (di default vale false)

Tabella 3.8 I metadati per la personalizzazione della annotazione @javax.jws.WebResult.

Ecco un esempio di applicazione della annotazione @WebResult e @WebParam per un ipotetico Web Service per la ricerca di un immobile in vendita:

```
@WebService
public class NuovoWS {
 @WebMethod
 @WebResult(name="Immobile")
 public Immobile cercaVendita(
  @WebParam(name="costo") double costo,
  @WebParam(name="DescrizioneComune") String comune,
  @WebParam(name="TipologiaCasa") TipoCasa casa)
 }
};
```

3.19 PERSONALIZZAZIONE SOAP

È possibile intervenire sul modo in cui verrà creato il messaggio soap grazie alla annotazione @javax.jws.soap.SOAPBinding. In **Tabella 3.8** i metadati utilizzabili per questa annotazione e il relativo significato.

3.20 CASO “SPECIALE”: SOLO INPUT

La annotazione @Oneway su un metodo indica che esso ha solo un messaggio di input e non ne ha di output. A differenza di un metodo “normale” in cui il tipo di ritorno è void, questa annotazione potrebbe far sì che il controllo al chiamante venga restituito subito dopo l'invocazione del metodo e non, come normalmente accade, quando la sua esecuzione termina (ma questo comportamento è contemplato dalla specifica, poi dipende dalla singola implementazione applicarla o meno).

Metadato	Significato
Style	Specifica lo stile da usare nell'encoding del messaggio. I due stili utilizzabili sono DOCUMENT (valore di default) e RPC. Quest'ultimo si chiama così in quanto è quello usato per le invocazioni Remote Procedure Call e prevede che il Body contenga un elemento il cui nome è quello della procedura remota da invocare. Invece se un messaggio ha il formato Document, allora il body contiene uno o più sotto-elementi, chiamati parti, ciascuna delle quali contiene documenti generici (il cui contenuto verrà formalizzato nel WSDL)
Use	Definisce lo stile di formattazione del messaggio. Può valere LITERAL (valore di default) o ENCODING. ENCODING è una specifica di SOAP 1.1 (resa obsoleta con la specifica SOAP 1.2) che indica come certi oggetti, strutture ed array (ma anche grafi rappresentati strutture di oggetti complesse) debbano essere serializzate. LITERAL demanda ad un XML Schema il compito di definire le regole per la serializzazione.
parameterStyle	Può valere BARE o WRAPPED (valore di default) e indicano come i parametri sono inseriti nel messaggio. Nel caso di WRAPPED c'è un unico elemento di più alto livello che fa da wrapper ai diversi parametri; nel caso di BARE ogni parametro compare allo stesso livello

Tabella 3.9 I metadati per la personalizzazione dei messaggi SOAP grazie alla annotazione @javax.jws.soap.SOAPBinding.

Attenzione
Per una migliore interoperabilità del Web Service è consigliato mantenere i valori di default o, per lo meno, DOCUMENT/LITERAL come, rispettivamente, stile e uso. L'altra modalità riconosciuta da WS-I (l'organismo che si occupa dell'interoperabilità dei Web Services, il cui sito di riferimento è http://www.ws-i.org/, Figura 3.8), è RPC/LITERAL. È meglio evitare le altre combinazioni che, pur possibili, non sono standard.

3.21 BASTA QUESTO?

È possibile, grazie a JAX-WS, creare un client in maniera statica o dinamica. Nel primo caso si tratta di costruire (o generare, grazie ad appositi tool) le classi che interagiscono con il Web Service e incapsulano la logica di interscambio dei dati. Nel caso di client dinamici non vengono generate le classi staticamente, ma vengono generati oggetti istanziati per uno specifico servizio e che incapsulano i meccanismo di comunicazione (in pratica nel caso di classi statiche è necessario ricrearle ad ogni modifica del WSDL; nel caso di client dinamici no).
Fin qui sono stati realizzati esempi di Web Services piuttosto semplici.

Figura 3.8: La home page della Web Services Interoperability Organization.

```
C:\WINDOWS\system32\cmd.exe

D:\IVAN\scritti\libri\NovitaDiJava6>apt it\ioprogr
it\ioprogrammo\java6\ws\secondoWS.java:15: The met
ng.String) of class it.ioprogrammo.java6.ws.second
as a return type.
  public boolean isPartitaIvaValida(String partita
                 ^
1 error

D:\IVAN\scritti\libri\NovitaDiJava6>_
```

Figura 3.9: Errore segnalato per un metodo @Oneway ma che restituisce un valore.

In realtà esistono numerose specifiche per la creazione di Web Services ben più complessi ed elaborati. Però, prima di addentrarci in questi dettagli, si potrebbe avere la curiosità di indagare i messaggi scambiati tra client e server e capire, a basso livello, che tipo di informazioni vengono scambiate e con quali formati…

3.22 UN MONITOR

Esistono diversi tool che permettono di creare dei monitor che si mettono in "ascolto" su una determinata porta e riportano le informazio-

Attenzione

Se si indica la annotazione @Oneway su un metodo che restituisce dei valori o che ha una o più eccezioni, si ha un messaggio di errore e fallisce la generazione della classe da parte del pre-processore (Figura 3.9):

```
@javax.jws.WebMethod
@javax.jws.Oneway
public boolean isPartitaIvaValida(String partitaIva){
  // algoritmo di verifica
           return true;
}.
```

Attenzione
Storicamente, uno dei framework più utilizzati in Java per la creazione di Web Services, è Axis (l'home page del progetto è http://ws.apache.org/axis/). Esso permette la creazione di parti server e client. Queste ultime, però, sono solo di tipo statico, ovvero appositi tool del framework permettono la creazione, una tantum, delle classi a partire da uno specifico WSDL. La nuova versione, Axis 2 (http://ws.apache.org/axis2/) supera anche questo limite.

ni che transitano; oltre ad eseguire il log inviano tali informazioni su una porta diversa (quella a cui, effettivamente, risponderà il servizio Web). Per esempio si supponga di avere un servizio in ascolto sulla porta 8080. Se il monitor accetta richieste sulla porta 8081, allora il client deve indicare 8081 come porta a cui inviare i dati. Il monitor li intercetta, li mostra (o su un file di log o su una finestra grafica) e li ri-trsmette sulla porta 8080. Il risultato viene a sua volta mostrato dal monitor e ripassato al client (in pratica il monitor fa da proxy). Avere uno di questi tool è fondamentale quando iniziano a presentarsi dei problemi nei Web Services tra un client ed un server. In effetti una situazione comune è che un server (scritto in un linguaggio o piattaforma diversa) viene documentato unicamente da un WSDL e, al più, da un messaggio SOAP di esempio. Ecco che analizzando il messaggio effettivamente inviato dal client aiuta a capire le differenze e tentare azioni di correzione mirate. Il tool che analizziamo è "JAX-WS WSMonitor Tool", disponibile per il download dalla home del progetto: https://wsmonitor.dev.java.net/. Viene scaricato un file .jar che è un semplice installer. Per eseguirlo:

```
> java -jar wsmonitor-installer-[versione].jar
```

Dopo aver accettato la licenza d'uso (i termini sono quelli di "COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL)"), vengono installati i file necessari (Figura 3.10). Per eseguire il monitor si può

```
C:\WINDOWS\system32\cmd.exe
D:\>java -jar wsmonitor-installer-1.1.jars

wsmonitor
wsmonitor\bin
wsmonitor\etc
wsmonitor\lib
wsmonitor\CDDLv1.0.1.txt
wsmonitor\bin\wsmonitor.bat
wsmonitor\bin\wsmonitor.sh
wsmonitor\etc\config.xml
wsmonitor\lib\FastInfoset.jar
wsmonitor\lib\args4j-2.0.6.jar
wsmonitor\lib\jsr173_api.jar
wsmonitor\lib\sjsxp.jar
wsmonitor\lib\wsmonitor.jar
wsmonitor\readme.txt
installation complete

D:\>
```

Figura 3.10: Installazione di "JAX-WS WSMonitor Tool".

eseguire wsmonitor\bin\wsmonitor.bat (se si usa Linux c'è un analogo file con estensione .sh). Di default il monitor resta in ascolto sulla porta 4040 e trasmette i dati alla porta 8080 (è possibile configurare il monitor affinché si metta in ascolto su più di una porta; eventuali configurazioni vanno specificate in un apposito file XML e specificato come primo argomento sulla linea di comando; un esempio di file di configurazione è presente in wsmonitor\etc\config.xml). Il monitor ha due

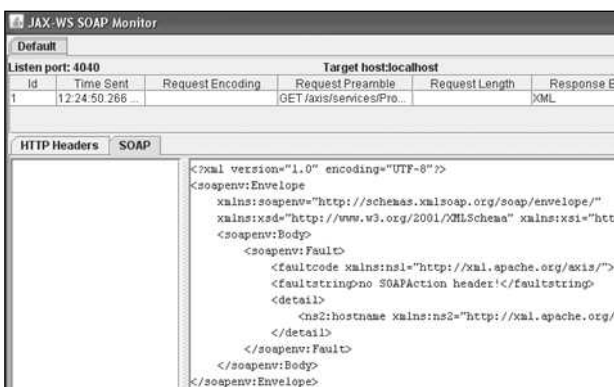



Figura 3.11: Log di esempio di "JAX-WS WSMonitor Tool".

tab: uno per le richieste http (ne mostra gli header) e uno per le richieste SOAP (con i dettagli dei messaggi). In Figura 3.11 un esempio d'uso.

3.23 USARE I WEB SERVICE IN NETBEANS

L'introduzione delle annotazione semplifica di molto la creazione di WebService. La cosa è ancora più evidente se si fa uso di un IDE: in questo caso, infatti, non c'è nemmeno la necessità di invocare tool separati (come wsimport, wsgen o apt) ma il tutto viene fatto all'interno dell'editor. Per i dettagli si può far riferimento alla documentazione ufficiale e, in particolare, alla pagina http://www.netbeans.org/kb/trails/web.html (c'è un'intera sezione dedicata ai Web Services). Alla pagina http://download.java.net/javaee5/screencasts/hello-simple-glassfish/ c'è un interessante video che mostra, passo a passo, la creazione di Web Services con l'ultima release di NetBeans, la 5.5.1.

Attenzione
Al momento dei test si sono verificati dei problemi usando il monitor con alcuni tipi di server (per esempio usando la classe eseguiWS illustrata in precedenza). Questo è dovuto con tutta probabilità alla relativa instabilità della versione 1.1 che, al momento dei test, non è dichiarata stable.

3.24 WS-SECURITY? NON USARE SOLO JDK 6!

Alcuni sostengono che è stato azzardato includere le tecnologie legate ai WS direttamente nel JDK visto che queste diventano facilmente obsolete. C'è però chi preferisce avere una versione standard, anche se obsoleta, che nessun supporto nativo. In realtà la versione JAX-WS 2.0,

quella includa nel JDK 6, è stata scelta unicamente perché l'unica completata in tempo per il rilascio del JDK. La 2.1 offre alcuni miglioramenti molto significativi (non solo: essa è risultata disponibile poco dopo l'uscita del JDK).In realtà, per le usuali applicazioni che si basano sui Web Services ma che fanno uso solo di servizi basati su SOAP (o, al più, di tipo REST) usare una o l'altra versione non rappresenta un grosso problema. Diverso il caso in cui si voglia usare standard avanzati quali WS-Security, WS Messaging e WS-Secure Conversation. Pertanto, senza voler entrare nel merito della diatriba "Web Services in Java 6, sì o no", si vedrà come mantenere il JDK aggiornato utilizzando JAX-WS 2.1 (o, quando saranno disponibili, alle successive versioni). Per ora resta valido un consiglio: chi volesse usare uno standard come WS-Security aggiorni subito la propria installazione con JAX-WS 2.1!

3.25 AGGIORNARE JAX-WS

Per aggiornare il sistema all'ultima versione è necessario collegarsi alla home page del progetto (Figura 3.12).

Per l'aggiornamento del JDK è necessario ricorrere al meccanismo di

Attenzione
WSIT (https://wsit.dev.java.net/) è il progetto specifico che si occupa dell'implementazione degli standard quali WS-Security, WS Messaging e WS-Secure Conversation nonché delle problematiche di interoperabilità. La separazione in sottoprogetti permette una migliore modularità e una suddivisione delle responsabilità che aiuta a far evolvere JAX-WS in maniera ordinata e pulita.

ridefinizione delle librerie "endorsed" (http://java.sun.com/javase/6/docs/technotes/guides/standards/). In pratica esistono due tipi di tecnologie aggiornabili: le API Endorsed (ne fanno parte le API di Corba e Xml SAX) e le tecnologie dette "standalone" (esse sono quelle riportate in Tabel-

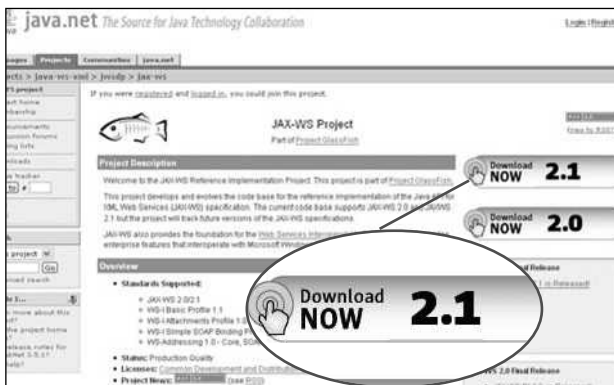



Figura 3.12: Home page del progetto JAX-WS; si noti l'indicazione per il download della versione 2.1.

la 3.10). In pratica è possibile aggiornare una qualsiasi di tali librerie con una versione più aggiornata, mantenendo le altre API del JDK (mentre non è possibile farlo per le API non definite come "endorsed"!). Per l'aggiornamento del JDK è necessario ricorrere al meccanismo di ridefinizione delle librerie "endorsed" (http://java.sun.com/javase/6/docs/technotes/guides/standards/). In pratica esistono due tipi di tecnologie aggiornabili: le API Endorsed (ne fanno parte le API di Corba e Xml SAX) e le tecnologie dette "standalone" (esse sono quelle riportate in Tabel-

Tecnologia (sigla), Versione di default in Java 6
Java API for XML Processing (JAXP), 1.4
Java Architecture for XML Binding (JAXB), 2.0
Java API for XML-Based Web Services (JAX-WS), 2.0
Java Compiler API, 1.0
Pluggable Annotation Processing API, 1.0
Common Annotations for the Java Platform, 1.0
Scripting for the Le tecnologie standalone in Java 6. Esso possono essere ridefinite usando le librerie endorsed.Java Platform, 1.0

Tabella 3.10

la 3.10). In pratica è possibile aggiornare una qualsiasi di tali librerie con una versione più aggiornata, mantenendo le altre API del JDK (mentre non è possibile farlo per le API non definite come "endorsed" !).
In pratica il meccanismo di ridefinizione funziona come segue:

- se esiste la proprietà di sistema java.endorsed.dirs, essa indica quali cartelle possono contenere file JAR che ridefiniscnono una o più librerie endorsed;

Attenzione
Il meccanismo di ridefinizione delle tecnologie endorsed è stato introdotto in Java 1.4. Però ogni versione del JDK/JRE possiede proprie librerie e API definite come endorsed. Far riferimento alla versione del JDK in uso per maggiori informazioni.

- se non esiste tale proprietà, vengono usati eventuali JAR presenti nella cartella $JAVA_HOME/lib/endorsed/ (dove con $JAVA_HOME si indica la cartella dov'è installato il JDK / JRE usato).

È anche possibile evitare di definire la proprietà di sistema e indicarla come parametro sulla linea di comando di invocazione del tool usato; per esempio:

```
java -cp . -Djava.endorsed.dirs=/path/lib/libreria.jar package.Classe
```

3.26 MIGLIORIE DI JAX-WS 2.1

Come spiegato in un blog degli sviluppatori (http://weblogs.java.net/blog/kohsuke/archive/2006/02/neckdeep_in_jax.html) la release 2.1 è una completa riscrittura della versione 2.0; questo ha comportato anche ad un'evoluzione architetturale non semplice. Però i risultati sono stati notevoli, sia in termini di miglioramento di performance che di una netta

separazione di responsabilità tra i componenti del sistema (che aiuta sia nel debug del codice che ad una sua evoluzione).
Tra le caratteristiche di JAX-WS 2.1 si segnala l'aderenza ai seguenti standard:

Web Services Addressing 1.0 - Core, SOAP Binding e WSDL Binding
WS-Addressing
SOAP versioni 1.1 e 1.2
REST e XML/HTTP
WS-I Basic Profile 1.1, Simple SOAP Binding Profile 1.0 e Attachment Profile 1.0
MTOM

Inoltre è possibile l'integrazione con il progetto WSIT, che comporta la compatibilità con la piattaforma .NET 3.0 e l'implementazione dei seguenti standard:

WS-ReliableMessaging
WS-Policy
WS-MEX
WS-Security/WS-Security Policy
WS-Trust/WS-Secure Conversation
WS-AT/WS-Tx (in questo caso è necessario Glassfish v2)
SOAP/TCP

Esistono anche tool per il passaggio da WSDL a classi Java (e viceversa) che possono essere usati sia come tool a linea di comando che come task Ant o plugin per Maven2. Ulteriori informazioni sulle caratteristiche della versione 2.1 possono essere trovate alla pagina http://weblogs.java.net/blog/vivekp/archive/2007/02/jaxws_21_fcs_fa_1.html.
Detto questo è il momento di passare ad analizzare le migliorie introdotte da Java 6 per la gestione dei documenti XML. Queste migliorie hanno un impatto anche sulla gestione dei WS, in quanto ne rappresentano le fondamenta.

3.27 LE TECNOLOGIE XML

In Tabella 3.9 tutti gli standard utilizzabili in Java per la gestione dei documenti XML (alcuni solo alla base per la gestione stessa dei Web Services). Ciascuno è "specializzato" per alcuni aspetti dello sviluppo di Web Services. Questo proliferare di tecnologie (che oltre a introdurre sigle implica altrettanti packages diversi!) fa capire come da una parte lo sviluppo di nuovi Web Services sia semplice, ma la comprensione approfondita di tutte le tecnologie coinvolte sia un compito "da specialisti".Ora si vedrà una delle nuove caratteristiche introdotte in Java 6: la gestione delle firme nei messaggi XML…

3.28 JAVA XML DIGITAL SIGNATURES API

La sicurezza dei Web Services spesso passa per la cifratura dei messaggi XML. Ma c'è un altro aspetto che sta emergendo come interessante

Standard	Utilizzo
JAXM	Java API for XML Messaging: permette di mandare e ricevere documenti XML; la specifica implementa Simple Object Access Protocol (SOAP) 1.1 Questo permette di focalizzarsi al livello del messaggio e non del corrispettivo XML
JAXB	Java Architecture for XML Binding; permette di passare da XML Schema ad un modello ad oggetti Java
JAXP	Java API for XML Processing: permette di accedere alle tecnologie di parsing e trasformazione dei documenti XML astraendo dalle caratteristiche di una particolare implementazione
JAXR	Java API for XML Registries: permette di interagire in maniera uniforme ad una serie di registri, alcuni basati su standard aperti (ebXML ne è un esempio) altri basati su standard riconosciuti e standardizzati (per esempio UDDI)

Tabella 3.9 Standard Java per la gestione dei documenti XML.

per le sue ripercussioni: la firma digitale dei documenti XML (firma che è stata recentemente riconosciuta, e incoraggiata, per i documenti ufficiali verso lo Stato Italiano). Infatti in numerosi contesti sono sempre più richieste caratteristiche quali non repudiabilità e certezza del mittente, conformità del messaggio rispetto al suo originale e così via. Per affrontare queste e altre problematiche Java ha predisposto una specifica JSR, la numero 105: Java XML Digital Signatures API. Il JDK 6 include il supporto per tale specifica. Ecco come usarla e i principi che regolano il suo utilizzo…

3.29 I TIPI DI FIRMA

In Java sono permessi tre tipi di firma digitale dei documenti XML: Enveloped Signature (firma imbustata, ovvero la firma sta dentro il documento XML, come ulteriore elemento), Enveloping Signature (firma imbustante, cioè essa racchiude il documento XML) e Detached Signature (firma separata: la firma è separata dal documento XML cui si riferisce). In **Figura 3.13** una rappresentazione delle tre tipologie. La tipologia che con più probabilità torna utile è la Enveloped. Questa ha, infatti, due ottime caratteristiche: da un lato è possibile avere a disposizione la firma direttamente nel documento XML, senza doverla reperire separatamente (cosa non possibile per la tipologia Detached) ma dall'altro lato permette di usufruire del contenuto del documento XML, anche se chi la usa non è in grado (o non è interessato) di leggere la firma (questa cosa è impossibile per la tipologia Enveloping: infatti per

Figura 3.13: I tre tipi di firma previsti in Java per i documenti XML: Enveloped, Enveloping e Detached

poter accedere al contenuto XML è necessario "togliere" la busta andando a interagire con la firma stessa).

3.30 ALGORITMI A CHIAVE PUBBLICA

Alla base delle implementazioni più diffuse per la firma digitale c'è un algoritmo di cifratura che permette di usare algoritmi di cifra detti a "chiave pubblica" (o PKC: Public Key Cryptography). Ogni entità (in particolare chi emette il documento) deve possedere una coppia di chiavi: una è privata, l'altra è resa pubblica. Le due chiavi sono correlate da algoritmo per il quale ci sono queste caratteristiche:

- la chiave privata deve essere segreta e tale deve essere mantenuta;
- la chiave pubblica è resa disponibile a chiunque;
- è possibile calcolare semplicemente la chiave pubblica a partire da quella privata; l'operazione inversa (calcolo della chiave privata a partire da quella pubblica) deve essere molto complessa tanto da risultare impraticabile (ovvero anche con un sistema molto potente e pur conoscendo l'algoritmo, è praticamente impossibile risalire al calcolo della chiave privata).

Inoltre deve essere disponibile un algoritmo (noto e documentato) per il quale l'utente che firma un messaggio applica la propria chiave privata per cifrarlo (tutto o, come vedremo, una sua parte) e solo con la chiave pubblica è possibile ottenere il documento originario a partire dal documento cifrato.

3.31 COSA SIGNIFICA "FIRMARE" UN DOCUMENTO

Firmare un documento significa:

calcolare l'impronta del documento; tale calcolo si basa su algoritmi

per cui una minima variazione del documento originario deve coincidere con una variazione della sua impronta (questa viene detta hash, visto che normalmente si applicano funzioni hash per il suo calcolo, o digest); Viene applicato un algoritmo di firma a chiave pubblica all'impronta; La firma viene, nel caso di documenti enveloped, allegata al documento come, di solito, pure la chiave pubblica.

“Verificare” un documento firmato

Una volta ottenuto un documeno firmato, chiunque può, avendo la chiave pubblica di chi dice di aver firmato il documento, verificarlo (il risultato è verifica valida o fallita); per procedere con la verifica è necessario:

Ottenere l'impronta del documento ricevuto;

Decrittare la firma del documento usando la chiave pubblica;

Verificare che le due impronte, una calcolata dal documento e una ottenuta dalla firma, coincidano. Se sì si può concludere che la firma è valida.

Che significa che è valida? Significa sia che siamo certi che chi ha eseguito la crittografia è colui la cui chiave pubblica è in nostro possesso sia che il documento che abbiamo verificato non è stato modificato da quando è stato firmato. In pratica abbiamo ottenuto sia la certezza del mittente che il fatto che il documento non è stato alterato successivamente. Non solo: solo chi è in possesso della chiave privata era in grado di apporre la firma; in pratica, c'è anche la non repudiabilità del documento.

3.32 FIRMARE UN DOCUMENTO CON JAVA

Si proverà ad applicare la firma ad un documento XML di prova; tale documento è un semplicissimo documento XML con alcune informazioni

su una ipotetica scheda di questo libro (file test.xml):

Attenzione

Un problema spinoso, e di non semplice soluzione, è la questione della scadenza dei certificati. Infatti, più per ragioni commerciali che pratiche, ogni azienda che rilascia certificati impone delle scadenze alla loro validità. Di solito sono dell'ordine di un anno solare. Una soluzione consiste nel riapplicare, con i nuovi certificati, la firma ai documenti già firmati.

```
<libro>
  <titolo>News in Java 6</titolo>
  <autore>Ivan Venuti</autore>
  <editore>Edizioni Master</editore>
  <pagine>160</pagine>
</libro>
```

Alla pagina http://java.sun.com/javase/6/docs/technotes/guides/security/xmldsig/overview.html c'è, oltre ad un'introduzione sulla tecnologia, una sezione di esempi; per semplicità si farà riferimento all'esempio "Generating an enveloped XML Digital Signature" (seguendo il link si esegue il download della classe GetEnveloped.java che, per comodità, è allegata ai sorgenti di questo libro). Come primo argomento il programma vuole il file di input; se esiste anche un secondo argomento questo viene usato per salvare il documento firmato, altrimenti lo stampa a video. Ecco come generare un documento (e salvarlo come firmato.xml) a partire dal nostro test.xml:

```
>java GenEnveloped test.xml firmato.xml
```

Il risultato è il seguente documento (uguale al precedente ma con in più la firma!):

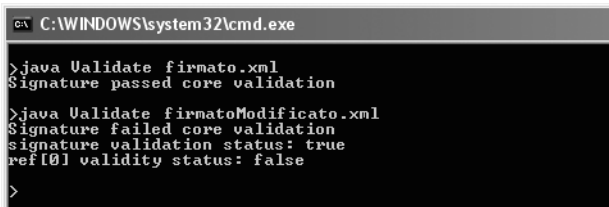
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><libro>
 <titolo>News in Java 6</titolo>
 <autore>Ivan Venuti</autore>
 <editore>Edizioni Master</editore>
 <pagine>160</pagine>
<Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
 <SignedInfo>
  <CanonicalizationMethod Algorithm=
  "http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments"/>
  <SignatureMethod Algorithm=
    "http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
  <Reference URI="">
    <Transforms>
     <Transform Algorithm=
       "http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
    </Transforms>
    <DigestMethod Algorithm=
    "http://www.w3.org/2000/09/xmldsig#sha1"/>
    <DigestValue>IPr1dbBaR6AKgnSPQWp/r/Q9t8k=</DigestValue>
  </Reference>
 </SignedInfo>
 <SignatureValue>
   iluB3U4xWDMMdW21t281zVL5MPIlWWTXCRHsWxNRAWfIZOD///1OGA==
 </SignatureValue>
 <KeyInfo>
  <KeyValue>
   <DSAKeyValue> <P>/KaCzo4Syrom78z3EQ5SbbB4sF7ey80etKII864WF64
 B81uRpH5t9jQTxeEu0ImbzRMqzVDZkVG9xD7nN1kuFw==</P><Q>li7dzDac
uo67Jg7mtqEm2TRuOMU=</Q><G>Z4Rxsnqc9E7pGknFFH2xqaryRPBaQ01k
 hpMdLRQnG541Awtx/XPaF5Bpsy4pNWMOHCBiNU0NogpsQW5QvnlMpA==
</G><Y>zi1Ie7jWeFtmj2cWgHn0w525Fva9NqH+IBj7zZ+9Kp7N/T9tlpLLxc3Q
                                    +lynqL2Ru4yrqbTYzOHgFVgfOUFcUQ==</Y>
   </DSAKeyValue>
```

```
 </KeyValue>
</KeyInfo>
</Signature>
</libro>
```

3.33 VALIDARE UN DOCUMENTO

Ottenuto il file firmato.xml, è possibile validarlo usando un altro degli esempi proposti nella guida segnalata in precedenza: Validate.java (scaricabile seguento il link "Validating an XML Digital Signature"). La sua sintassi è:

```
>java Validate nomeDocumentoDaValidare.xml
```

In Figura 3.14 due invocazioni: una su firmato.xml ottenuto dal programma di firma visto in precedenza (validazione andata a buon fine) e una su firmatoModificato.xml che è lo stesso documento ma con il valore 161 su pagine anziché 160 come nel documento originario. In quest'ultimo caso la firma non risulta più valida!

3.34 ACCESSO ALLE RISORSE NATIVE PER LA SICUREZZA

Gli algoritmi di cifratura e di decifratura sono particolarmente onerosi in termini di risorse. Per questo è importante, per contesti "critici", po-

Attenzione

Per renderlo "comprensibile", il file generato è stato formattato inserendo opportuni "a capo" e indentando i diversi tag. Questo ha però reso invalida la firma! Infatti anche un solo nuovo spazio di differenza fa sì che l'impronta generata non sia più la stessa (e di conseguenza nemmeno la firma).

ter accedere alle risorse native per la gestione della sicurezza. Tali risorse possono includere acceleratori hardware dedicati ma anche infrastrutture specifiche per i diversi sistemi operativi. Un esempio è l'architettura Microsoft CryptoAPI presente sui sistemi Windows. Attraverso questa infrastruttura si può accedere sia ad operazioni di cifra che ai certificati gestiti dal sistema operativo. Java 6 offre un accesso diretto a queste infrastrutture e permette di usarle all'interno dei propri programmi grazie a nuove API. Un buon articolo che ne introduce caratteristiche e potenzialità è "Levering Security in the Native Platform Using Java SE 6 Technology", reperibile sul sito della Sun alla pagina http://java.sun.com/developer/technicalArticles/J2SE/security/.

```
C:\WINDOWS\system32\cmd.exe

>java Validate firmato.xml
Signature passed core validation

>java Validate firmatoModificato.xml
Signature failed core validation
signature validation status: true
ref[0] validity status: false

>
```

Figura 3.14: Verifica della firma su due documenti. Solo il primo ha una firma valida.

3.35 CONCLUSIONI

L'uso di XML è sempre più traversale e diffuso sia come interscambio di dati ma anche come formato per la memorizzazione di file di configurazione e come base per architetture distribuite basata sul concetto di Web Service. Java 6 offre numerosi e avanzati strumenti per la gestione completa di tutte le problematiche, dall'accesso e gestione dei documenti fino ad annotazioni di alto livello che incapsulano tutta la logica e la complessità dell'esporre una classe come servizio Web. Come si vedrà nel prossimo capitolo, XML entra con prepotenza anche nella

gestione dei dati da e verso basi di dati, con nuovi tipi di dato e nuove operazioni che gestiscono documenti XML in maniera nativa. È venuto il momento di presentare JDBC 4.0!

JDBC 4.0

Java permette l'accesso alle basi dati grazie ad un set standard di API, definite dallo standard JDBC (secondo alcuni sarebbe l'acronimo di Java DB Connection, ma la Sun ha sempre negato che JDBC fosse un acronimo o avesse un qualche significato preciso!). Chi implementa lo standard è un componente software chiamato "driver JDBC": esso implementa le diverse API per accedere in maniera appropriata alla risorsa (in questo senso è perfettamente possibile che la risorsa non sia un DBMS, anche se il caso più frequente è l'accesso a basi di dati relazionali).
In Java 6 sono state introdotte interessanti novità, tanto che la versione dello standard è diventata la 4.0. Nel loro insieme queste funzionalità mirano a rendere più semplice lo sviluppo di software che si connette alle basi dati. Però per apprezzare le novità nel loro insieme, si farà una panoramica su JDBC (dove verranno evidenziate alcune novità di base, nel seguito saranno mostrate le novità più "evolute").
È bene precisare che la nuova versione dello standard resta compatibile con il codice scritto secondo le versioni precedenti. In questo modo è assicurata la compatibilità all'indietro di tutto il software esistente per l'accesso alle basi dati.

4.1 JDBC "MINIMALE"

JDBC è uno strato software che ha come fine ultimo la scrittura di codice indipendente dal database effettivamente utilizzato. In pratica ogni database deve, per essere usato con questa tecnologia, disporre di un proprio driver JDBC.
Un driver non è altro che un componente che implementa un insieme di API standard. In questo modo chi scrive il codice Java fa uso di API generiche, quelle dello standard JDBC per l'appunto: cambiando driver nessuna modifica è necessaria al resto del software. Di seguito una panoramica delle classi del package java.sql.

4.2 UN DBMS PER INIZIARE

È possibile installare un qualsiasi database che fornisca driver JDBC. I più comuni DBMS (Oracle, MySql, Informix, …) sono adatti allo scopo. Però, per semplificare la creazione dell'ambiente di sviluppo, si può far uso anche del DBMS chiamato Apache Derby. Esso è installato di default insieme al JDK 6.0. La sua caratteristica principale è di essere un DBMS caricato interamente in memoria (maggiori informazioni su tale DBMS sono reperibili alla home del progetto: http://db.apache.org/derby/). La versione del dbms che accompagna il JDK è chiamata Java DB (si veda anche la pagina http://developers.sun.com/prodtech/javadb/ per le informazioni specifiche al db allegato al JDK).

Attenzione

Per verificare la presenza del DB Apache Derby è sufficiente andare nella carte db/ sotto la cartella principale ove si è installato il JDK 6.0. Qui sono presenti sia i jar che la documentazione sulla versione corrente (da essa, per esempio, si viene a conoscere che la versione che è inclusa nel JDK è la versione 10.2.1.7).

4.3 CONNETTERSI ALLA SORGENTE DATI: OGGETTO CONNECTION

Un oggetto di tipo Connection rappresenta una connessione alla sorgente dati ed è l'oggetto principale per poterne utilizzare i dati e inviare istruzioni di modifica e aggiornamento dei dati. Una connessione viene reperita dal driver. In passato, prima di far uso di un qualsiasi driver JDBC, era necessario invocare per nome la classe; per questo è comune trovare nel codice una simile istruzione:

```
Class.forName(driver).newInstance();
```

Con JDBC 4.0 questo non è più necessario: la prima volta che l'applicazione tenta una connessione alla sorgente dei dati, la classe DriverMa-

nager caricherà (in maniera automatica!) qualisasi driver JDBC presente nel suo CLASSPATH:

```
connessione =
 DriverManager.getConnection(
  "url.specifica.per.connessione"
 );
```

Per questo motivo è sufficiente includere i driver specifici per il DBMS in uso per far funzionare l'applicazione con diversi DBMS senza più l'onere di configurare opportunamente l'ambiente. Per verificarlo è sufficiente eseguire questo metodo:

```
public void listaDriver(){
 java.util.Enumeration<Driver> tuttiDriver =
   DriverManager.getDrivers();
 while(tuttiDriver.hasMoreElements()){
   Driver driver = tuttiDriver.nextElement();
   System.out.println("<trovato driver> = "+
     driver.getClass().getName());
 }
}
```

Prima da una classe nel cui classpath non è specificato alcun driver aggiuntivo:

```
>java -classpath
 c:\Programmi\Java\jdk1.6.0\db\lib\derby.jar;%CLASSPATH%
 it.ioprogrammo.java6.db.AccessoJavaDB
```

L'unica stampa del metodo è:

```
<trovato driver> = sun.jdbc.odbc.JdbcOdbcDriver
```

Poi aggiungendo il driver incluso nel JDK 1.6:

```
>java -classpath
 c:\Programmi\Java\jdk1.6.0\db\lib\derby.jar;%CLASSPATH%
 it.ioprogrammo.java6.db.AccessoJavaDB
```

In questo caso c'è un secondo driver caricato:

```
<trovato driver> = sun.jdbc.odbc.JdbcOdbcDriver
<trovato driver> = org.apache.derby.jdbc.AutoloadedDriver
```

Accanto a DriverManager è possibile usare un oggetto DataSource per reperire una connessione. Spesso si preferisce l'uso di DataSource a DriverManager perché la sua definizione viene fatta esternamente all'applicazione (di solito fatta a livello di container, per esempio a livello di Servlet Container, per le applicazioni Web).
Esistono anche due estensioni a DataSource:

- **ConnectionPoolDataSource:** ha un supporto al riuso delle connessioni; questo migliora le prestazioni in quanto l'ottenimento di una nuova connessione è quasi sempre un'operazione lenta e dispendiosa in termini di risorse;
- **XADataSource:** permette di reperire connessioni che possono partecipare a transazioni distribuite (caratteristica auspicabile, e in certi contesti indispensabile, in caso di diverse sorgenti di dati).

4.4 COMANDI VERSO IL DB

Attraverso un oggetto Statement è possibile inviare dei comandi al Db. Ecco, per esempio, due routine che permettono, rispettivamente, la creazione e la distruzione di una tabella:

```
public void createDb(Connection conn) throws SQLException{
```

```
  Statement comando = conn.createStatement();
  comando.execute("create table articoli("+
    "chiave int NOT NULL GENERATED ALWAYS AS IDENTITY " +
    "   CONSTRAINT WISH_PK PRIMARY KEY,"+
    " descr varchar(80),"+
    " autore varchar(30) )");
  comando.execute("insert into articoli(descr,  autore)"+
   " values ('Usare Cayenne come tecnologia di persistenza',"+
   " 'Ivan Venuti')");
  comando.execute("insert into articoli(descr,  autore)"+
   " values ('JBoss Seam', 'Ivan Venuti')");
  comando.execute("insert into articoli(descr,  autore)"+
   " values ('Altro articolo', 'Altro autore')");
  comando.close();
}

public void destroyDb(Connection conn) throws SQLException{
  Statement comando = conn.createStatement();
  comando.execute("drop table articoli");
  comando.close();
}
```

Si noti che ogni comando viene chiuso; benché Java lo faccia in automatico è buona norma darlo non appena il comando non viene più usato.
In alcuni casi i comandi prevedono dei parametri. È sconsigliato includere i valori facendo la concatenazione tra le stringhe (si pensi ai problemi legati agli escape dei caratteri!), ma è sempre bene usare un altro tipo di comando: PreparedStatement. Ecco, per esempio, come cercare per autore:

```
PreparedStatement query = conn.prepareStatement(
  "SELECT chiave, descr, autore"+
```

```
 " FROM articoli"+
 " WHERE autore=?"+
 " ORDER BY descr");
query.setString(1, "Ivan Venuti");
```

4.5 L'INTERFACCIA RESULTSET

I risultati di una query SQL possono essere acceduti da un oggetto ResultSet:

```
public void performSql(Connection conn) throws SQLException{
  PreparedStatement query = conn.prepareStatement(
    "SELECT chiave, descr, autore"+
    " FROM articoli"+
    " WHERE autore=?"+
    " ORDER BY descr");
  query.setString(1, "Ivan Venuti");
  ResultSet rs = query.executeQuery();
  while(rs.next())
    System.out.println( "Trovato"+
       "articolo '"+rs.getString("descr")+"', "+
       "chiave="+rs.getString("chiave") );
  rs.close();
  query.close();
}
```

Si pensi ad un ResultSet come all'insieme di righe di una query. Con next() si passa alla riga successiva; per reperire gli elementi di una riga si invoca get"tipoDato". Il parametro di questa get può essere un nome (nome della colonna) o la posizione (rispetto ai diversi parametri restituiti).

4.6 L'INTERFACCIA ROWSET

Usando un ResultSet è necessario essere connessi alla sorgente dei dati. Per lavorare in maniera disconnessa lo standard JDBC permette di usare l'interfaccia RowSet. Essa può essere vista come un contenitore di dati espressi in forma tabulare che può essere serializzato e, pertanto, può essere inviato a client remoti. Un suo writer personalizzato permette di mandare gli eventuali aggiornamenti ai suoi dati in maniera che tali modifiche possono essere rese persistenti sulla sorgente dati di provenienza. Per i dettagli si può far riferimento al documento "JDBC RowSet Implementations Tutorial" alla pagina http://java.sun.com/j2se/1.5/pdf/jdbc-rowset-tutorial-1.5.0.pdf.

4.7 JDBC 4.0: CARATTERISTICHE AVANZATE

Accanto alle novità già introdotte, JDBC 4.0 fornisce un nuovo tipo di eccezioni che possono essere "in cascata". Inoltre introduce il concetto di identità di riga attraverso il supporto ai RowId e supporto per XML secondo il nuovo standard SQL2003.
Ecco i dettagli

4.8 ROWID

Per quanto riguarda le basi di dati relazionali si è soliti pensare all'identificazione di un record dalla sua chiave primaria. RowId rappresenta un indirizzo logico verso un determinato record (tale indirizzo può essere sia logico che fisico). A differenza di una chiave, un RowId ha una validità che è data dalla validità specificata dal data source o finché il record non viene eliminato. In Tabella 4.1 sono elencate le validità permesse e restituite dal metodo DatabaseMetaData.getRowIdLifetime(). Con la versione di default in Java 6 di Apache Derby, si ottiene un ROWID_UNSUPPORTED. Per verificarlo basta eseguire le seguenti istruzioni:

Costante	Significato
ROWID_UNSUPPORTED	Caratteristica non supportata dal DBMS
ROWID_VALID_OTHER	Tempo di validità non conosciuto per questa implementazione
ROWID_VALID_ TRANSACTION	Valida sicuramente all'interno della transazione
ROWID_VALID_SESSION	È garantita la validità all'interno della sessione
ROWID_VALID_FOREVER	Non ha scadenza (ovvero vale fintantoché la riga non viene eliminata fisicamente dalla base dati)

Tabella 4.1 Tempo di validità del RowId.

```
System.out.println(" RowIdLifetime: "+
 conn.getMetaData().getRowIdLifetime().toString());
```

Un oggetto java.sql.RowId può essere ottenuto da un ResultSet oppure da un CallableStatement:

```
java.sql.RowId mioRowId = rs.getRowId(1);
```

L'oggetto ottenuto è immutabile e può essere usato per riferirsi in maniera univoca all'intera riga.

Attenzione
L'oggetto RowId è specifico per il data source usato e, come tale, non può essere usato su connessioni ottenute da datasource diverse!

4.9 SUPPORTO XML

JDBC 4.0 supporto l'XML come tipo di dato predefinito. Questo significa che si possono leggere/scrivere dati direttamente nel formato XML. Ovviamente questo supporto è contemplato a livello di standard JDBC,

ma affinché tutto funzioni anche il driver utilizzato deve implementare queste caratteristiche e, ovviamente, il database sottostante deve prevedere questa nuova caratteristica (che, si ricorda, fa parte del nuovo standard SQL 2003). Il supporto al nuovo tipo di dato si concretizza nella possibilità di leggere/scrivere documenti XML. In pratica è possibile partire da un documento XML e creare un opportuno parser per la sua gestione. I parser che le implementazioni dovrebbero supportare (come minimo) sono DOM, SAX e StAX.

4.10 LEGGERE DATI XML

Un dato per essere letto deve, come sempre, far riferimento ad un ResultSet facendo riferimento o al nome della colonna o alla posizione all'interno de

```
SQLXML sqlXml = resultSet.getSQLXML(nomeCol);
```

Sul nuovo oggetto è possibile ottenere uno stream usando:

```
InputStream binStr = sqlXml.getBinaryStream();
```

A questo punto è possibile far riferimento ad una delle tante tecnologie di parser disponibili per la lettura vera e propria dei dati; ecco come usare un parser di tipo DOM (che costruisce una rappresentazione completa del documento in memoria; sulla rappresentazione è possibile navigare a piacimento):

```
DocumentBuilder parser =
  DocumentBuilderFactory.newInstance().newDocumentBuilder();
Document result = parser.parse(binStr);
```

In maniera analoga si può usare un parser di tipo SAX (supponendo che si sia scritta una classe, qualeHandler, che ne implementa l'handler):

```
SAXParser parser =
  SAXParserFactory.newInstance().newSAXParser();
parser.parse(binStr, qualeHandler);
```

Oppure un parser di tipo StAX:

```
XMLInputFactory factory = XMLInputFactory.newInstance();
XMLStreamReader streamReader =
  factory.createXMLStreamReader(binStr);
```

Leggendo le API della classe SQLXML si viene messi in guardia che potrebbe essere preferibile evitare la serializzazione su uno stream, come fatto poc'anzi; potrebbe risultare più efficiente ricorrere alla rappresentazione XML usando il metodo getSource(): come parametro al metodo va passata una classe che corrisponde al tipo di parser che si vuole usare. Per un documento DOM:

```
DOMSource domSource = sqlXml.getSource(DOMSource.class);
Document document = (Document) domSource.getNode();
```

Nel caso di parser SAX:

```
SAXSource saxSource = sqlXml.getSource(SAXSource.class);
XMLReader xmlReader = saxSource.getXMLReader();
xmlReader.setContentHandler(qualeHandler);
xmlReader.parse(saxSource.getInputSource());
```

Ecco come può essere gestito con StAX:

```
StAXSource staxSource =
 sqlxml.getSource(StAXSource.class);
XMLStreamReader streamReader =
 staxSource.getXMLStreamReader();
```

è anche possibile recuperare valori usando trasformazioni XSLT applicandole al valore del XML reperito:

```
File xsltFile = new File("qualeTrasformazione.xslt");
File risultatoXml = new File("risultato.xml");
Transformer xslt = TransformerFactory.newInstance().
  newTransformer(new StreamSource(xsltFile));
Source source = sqlxml.getSource(null);
Result result = new StreamResult(risultatoXml);
xslt.transform(source, result);
```

Tra le possibilità offerte c'è anche quella di poter usare interrogazioni XPath direttamente sui valori XML. In questo caso:

```
XPath xpath = XPathFactory.newInstance().newXPath();
DOMSource domSource = sqlxml.getSource(DOMSource.class);
Document document = (Document) domSource.getNode();
String expression = "/radice/percorso/@valore";
String barValue = xpath.evaluate(expression, document);
```

4.11 SCRIVERE DATI XML

Nel seguito alcuni frammenti di codice per scrivere dati XML usando una delle tipologie di parser disponibili. Nel caso DOM:

```
DOMResult ris = sqlXml.setResult(DOMResult.class);
ris.setNode(nomeNodo);
```

Per SAX:

```
SAXResult ris = sqlXml.setResult(SAXResult.class);
ContentHandler handler =
  ris.getXMLReader().getContentHandler();
```

```
handler.startDocument();
 // ... esegue il set di tutti gli elementi
handler.endDocument();
```

Infine, ecco il codice da scrivere per settare un contenuto usando StAX:

```
StAXResult ris = sqlXml.setResult(StAXResult.class);
XMLStreamWriter streamWriter = ris.getXMLStreamWriter();
```

4.12 QUALCHE TEST?

Come eseguire i test? Qui il punto dolente: al momento di scrivere la versione di Derby distribuita insieme al JDK 6.0 non supporto il nuovo tipo di dato. Infatti se si prova ad eseguire le righe di codice sopra riportate si ottiene un laconico:

```
java.sql.SQLFeatureNotSupportedException: Funzione non implementata:
                                                   nessun dettaglio.
```

Bisognerebbe attendere una versione di Derby con pieno supporto per questa nuova (ed estremamente interessante!) caratteristica. Alternative? Usare campi CBlob!

4.13 USO DI CBLOB PER DATI XML

In Java 6 il db Derby gestisce i campi XML; è il driver a non implemen-

Attenzione
Un oggetto SQLXML deve essere implementato per intero o per nulla; in questo senso si può essere tranquilli riguardo alle future implementazione dell'oggetto: o esso viene supportato appieno oppure si ha subito un'eccezione non appena si tenta di recuperarne un'istanza.

tare SQLXML. Ecco un modo per scrivere dei dati XML passando per un campo CBlob (per informazioni sui tipi di dato usabili in Derby si veda http://db.apache.org/derby/docs/10.2/ref/crefsqlj31068.html). Ecco la tabella con cui si eseguiranno le prove: accanto ad una chiave autoincrementante, si userà un campo di tipo XML:

```
Statement s = conn.createStatement();
s.execute("CREATE TABLE testXml("+
 "chiave int NOT NULL GENERATED ALWAYS"+
 " AS IDENTITY CONSTRAINT MY_PK PRIMARY KEY, "+
 "datiXml XML)");
s.close();
```

Specificato un qualche XML da scrivere, si può usare setClob a cui si passa uno StringReader aperto sull'XML da scrivere:

```
String qualeXml =
  "<test>"+
  "<titolo>prova</titolo>"+
  "<altro>nulla</altro>"+
  "</test>";
PreparedStatement p = conn.prepareStatement(
  "INSERT INTO testXml (datiXml) VALUES "+
  "(XMLPARSE (DOCUMENT CAST (? AS CLOB) PRESERVE WHITESPACE))");

p.setClob(1, new java.io.StringReader(qualeXml));
p.execute();
p.close();
```

Si noti come la stringa SQL del PreparedStatement esegue un parsing sull'oggetto inviato. Ecco come eseguire una lettura sui dati inseriti:

```
s = conn.createStatement();
```

```
rs = s.executeQuery(
  "SELECT XMLSERIALIZE (datiXml as CLOB) FROM testXml");
while( rs.next() ){
  java.io.Reader rd = rs.getClob(1).getCharacterStream();
  char[] buf = new char[256];
  while((rd.read(buf))>0)
    System.out.print( new String(buf) );
    System.out.println();
  }
rs.close();
s.close();
```

Attenzione
Se si dovesse presentare una eccezione del tipo "Failed to locate 'Xalan' API or implementation classes" significa che manca la libreria di parsing XML, Xalan per l'appunto. Il suo download può essere fatto a partire dalla pagina http://xml.apache.org/xalan-j/ .

4.14 NCLOB

NCLOB è un tipo di dato predefinito SQL per memorizzare CLOB (Character Large Obejct) usando il National Character Set. In java 6 è stato introdotta l'interfaccia java.sql.NClob (che estende Clob) per interagire con questo tipo di dati nei DB che lo supportano. Allo stesso modo del tipo Clob ha validità all'interno della trasazione in cui è stato creato. Il metodo getNClob ne permette il reperimento a partire da un ResultSet, CallableStatement o PreparedStatement. In maniera analoga esiste il metodo setNClob per aggiornare il valore sul DB.

4.15 PROPRIETÀ DEL DATABASE

L'oggetto java.sql.Connection ha nuovi metodi. Tra questi c'è getClien-

tInfo che permette di recuperare informazioni dal database. Il driver può memorizzare queste informazioni o in uno speciale registro, come parametro di sessione o tabella di sistema (l'implementazione specifica non è necessario conoscerla).
Le informazioni usuali che un database può restituire sono:

- **ApplicationName**: nome dell'applicazione che sta usando la connessione;
- **ClientUser**: nome dell'utente che sta usando la connessione (che può essere diverso dall'utente che l'ha aperta!);
- **ClientHostname**: nome del computer dove risiede l'applicazione che usa la connessione;

siccome queste proprietà non sono standard, per conoscere le proprietà specifiche del bdms in uso e dei suoi vincoli si può usare il metodo DatabaseMetaData.getClientInfoProperties. Tale metodo restituisce un ResultSet con le informazioni riportate in Tabella 4.2 (ordinate per no-

Colonna	Tipo	Significato
NAME	String	Nome della proprietà
MAX_LEN	int	Lunghezza massima del valore assegnabile alla proprietà
DEFAULT_VALUE	String	Valore assunto di default
DESCRIPTION	String	Descrizione (per esempio può contenere informazioni su dov'è memorizzata questa proprietà nel DBMS)

Tabella 4.2 Informazioni sulle proprietà supportate dal database in uso.

me). Ecco, per esempio, come stampare le caratteristiche delle proprietà del database in uso:

```
Connection conn = db.getMyConnection();
```

```
ResultSet rs = conn.getMetaData().getClientInfoProperties();
while( rs.next() ){
  System.out.println("NAME :"+rs.getString("NAME"));
  System.out.println("MAX_LEN :"+rs.getInt("MAX_LEN"));
  System.out.println("DEFAULT_VALUE :"+rs.getString("DEFAULT_VALUE"));
  System.out.println("DESCRIPTION :"+rs.getString("DESCRIPTION"));
}
rs.close();
```

Invece ecco come stampare i valori attuali delle proprietà:

```
Properties p = conn.getClientInfo();
if (p.isEmpty())
  System.out.println("Nessun risultato da getClientInfo");
Enumeration enumProp = p.keys();
while(enumProp.hasMoreElements()){

  String pr = (String) enumProp.nextElement();
  System.out.println( pr + ": "+ p.getProperty(pr));
}
```

È anche possibile settare una o più proprietà grazie al metodo setClientInfo:

```
conn.setClientInfo(nome, valore)
```

Nel caso di errore viene sollevata una eccezione di tipo SQLClientInfoException. Anziché aggiornare una proprietà alla volta si può costruire un opportuno oggetto Properties e passarlo come argomento:

```
Properties p = new Properties();
// Inizializza p
conn.setClientInfo(p);
```

Si noti che l'aggiornamento di una o più proprietà può essere differita fino alla prossima istruzione eseguita sul db (per rendere maggiormente efficiente l'operazione).

4.16 ECCEZIONE? TANTE E CON PIÙ CAUSE!

Da sempre, quando l'interazione con la base dati fallisce, viene sollevata una eccezione di tipo SQLException. Purtroppo in passato in molte occasioni comprendere la vera causa dell'eccezione era problematico, in quanto il messaggio era poco significativo (perché di solito il messaggio era troppo generale). Con JDBC 4.0 tale tipo eccezione può essere sia multipla (o in catena). Essa può avere più eccezioni: per ogni eccezione si può risalire alla successiva usando il metodo getNextException. Non solo: ogni l'eccezione può avere delle "cause" (anche in questo caso il loro numero può essere arbitrario!). Per risalire alla sua causa (se esiste!) si può usare il metodo getCause. Ecco due metodi che, insieme, possono stampare l'intera gerarchia:

```
public void printExceptions(SQLException e, boolean printCauses) {
  while(e!=null) {
    System.out.println("eccezione:"+e);
    if (printCauses)
      printCauses(e);
    e = e.getNextException();
  }

}

Public void printCauses(SQLException e) {
  Throable th = e.getCause();
  while(th!=null) {
    System.out.println("   causa:"+th);
```

```
    th = th.getCause();
  }
}
```

Ecco come potrebbe risultare un gestore di eccezione che stampa le gerarchia usando i metodi appena definiti:

```
catch(SQLException sqlEx){
  printExceptions(sqlEx, true);
}
```

Attenzione
Non tutti i DBMS supportano le eccezioni multiple. È ovvio che per chi non le supporta, questa caratteristica non può essere usata né dai driver JDBC né dai programmi Java che ne fanno uso.

4.17 SI PUÒ RIPROVARE? FORSE!

L'uso delle eccezioni e dei gestori è da sempre un argomento fonte di controversie: in teoria si dovrebbero gestire ad un livello in cui "si sa cosa fare" o, detto in altri termini, al livello minimo in cui ha senso prevedere delle azioni correttive per evitare l'eccezione. Ovviamente non sempre è possibile definire tali azioni correttive, anzi: il più delle volte è quasi impossibile, ma spesso si usano dei blocchi try/catch in maniera troppo "grossolana" per cui spesso l'unica vera azione è quella di notificare all'utente che qualcosa non è andato a buon fine e di fare uno stack trace dell'eccezione sui file di log. In JDBC 4.0 si è cercato di raffinare il meccanismo delle eccezioni suddividendole in due categorie: la prima (più vicina alle "solite" eccezioni) è detta di eccezioni non transitorie, la seconda di eccezioni transitorie. Queste ultime sono quelle eccezioni in cui è possibile che il solo rieseguire l'operazione possa portare ad un ri-

Classe	Significato
SQLFeatureNotSupported Exception	È stata specificata una caratteristica non supportata dal DBMS
SQLNonTransient ConnectionException	Connessione fallita
SQLDataException	Errore nei dati
SQLIntegrityConstraint ViolationException	Violato uno dei vincoli di integrità del db
SQLInvalidAuthorization Exception	Credenziali non valide
SQLSyntaxErrorException	Errore di sintassi

Tabella 4.3: Classi che notificano eccezioni non transitorie.

sultato senza errori. Viceversa, quelle non transitorie sono eccezioni che sicuramente, anche ritentado l'operazione, se non si è intervenuti o con nuovi dati o con nuove configurazioni, sono destinate a rifallire. Questa distinzione è definita nelle specifiche SQL 2003. In Java la distinzione viene fatta usando diverse sottoclassi, ognuna figlia di SQLException: ogni sottoclasse appartiene ad una o all'altra categoria. Nelle Tabelle 4.X e 4.x1 sono riportate le diverse sottocalssi.

4.18 CONCLUSIONI

Lo standard JBC 4.0 non è supportato appieno dai driver presenti di default nel JDK 6. Questo ha un impatto minimo in quanto i driver spe-

Classe	Significato
SQLTransientConnection Exception	Connessione fallita
SQLTransactionRollback Exception	Errore durante una rollback
SQLTimeoutException	Operazione non terminata nei tempi massimi previsti

Tabella 4.4: Classi che notificano eccezioni transitorie.

cifici del DB in uso saranno comunque aggiornati e installati insieme alle applicazioni (o a livello di container). Però desta qualche dubbio l'opportunità di fornire uno strumento come Apache Derby quando lo stato di sviluppo dei suoi driver risulta lontano dal pieno supporto delle nuove specifiche JDBC.

SCRIPTING

Java 6 offre interfacce standard per l'utilizzo di linguaggi di scripting nelle proprie applicazioni. Di default viene fornito supporto a JavaScript, grazie all'implementazione Rhino di Mozilla. In questo capitolo verranno mostrate le interfacce standard e il loro utilizzo per interagire con Rhino.

5.1 SCRIPTING… PERCHÉ?

La prima e più ovvia domanda è: quando c'è bisogno di un linguaggio di scripting nelle proprie applicazioni? I motivi possono essere vari; il più comune è fornire un meccanismo di configurazione delle proprie applicazioni fornendo un linguaggio di alto livello con una notevole potenza espressiva, come può essere JavaScript per l'appunto. Molti linguaggi di scripting sono pensati con caratteristiche che ne agevolano l'apprendimento e l'uso; non a caso JavaScript è considerato un linguaggio di programmazione semplice. Inoltre si potrebbero creare degli script per la realizzione di prototipi o piccoli test (per esempio di accettazione). Altri motivi potrebbero essere la creazione di script standard usabili in più contesti. Si pensi al caso di applicazioni Web, realizzate con tecnologia DHTML (ovvero HTML e JavaScript): è comune avere degli script per il controllo sintattico/semantico dei valori inseriti nelle form (date valide, indirizzi di email ben formati e così via). Usando un motore di scripting si potrebbe usare la stessa libreria anche per la validazione dei campi di un'applicazione grafica stand-alone realizzata con Java. Ma aldilà di questo è bene precisare che all'interno dei linguaggi di script supportati è possibile interagire con gli oggetti Java dell'applicazione stessa. In qualche modo un utente "avanzato" potrebbe inserire propri script per personalizzare il comportamento stesso l'applicazione.

5.2 USARE GLI SCRIPT

All'interno di un'installazione del JDK è presente, come si è detto, un motore di scripting per JavaScript. Tale motore deve essere istan-

ziato. Siccome in teoria ci potrebbe essere un qualsiasi numero di diversi motori, ciascuno deve essere referenziato secondo una qualche sua caratteristica. Il modo più semplice è ricercarlo per nome, grazie al metodo getEngineByName della classe ScriptingEngineManager (tutte le classi usate in questo capitolo, dove non è specificato diversamente, sono del package javax.script):

```
ScriptEngineManager engManager =
 new ScriptEngineManager();
ScriptEngine engine=
 engManager.getEngineByName(nome);
```

Come sapere quali engine ci sono a disposizione e qual è il loro nome? Un modo è reperirne la lista usando il metodo getEngineFactories. Ecco un semplice modo per stampare sulla console sia il nome dell'engine che il linguaggio supportato:

```
ScriptEngineManager scriptManager =
   new ScriptEngineManager();
java.util.List<ScriptEngineFactory> tuttiEngine =
  scriptManager.getEngineFactories();
for (ScriptEngineFactory engineAttuale : tuttiEngine) {
  System.out.println(
    "Engine " + engineAttuale.getEngineName()+ " - " +
    "linguaggio:" + engineAttuale.getLanguageName());
}
```

Verificando la stampa, si nota che in Java 6 c'è unicamente l'engine "Mozilla Rhino" per il linguaggio "ECMAScript". È interessante che il metodo getEngineByName vuole specificato il nome del linguaggio, non dell'engine, come potrebbe sembrare. Il nome che si può usare per Rhino è sia quello "ufficale", ovvero "ECMAScript", sia "js" che "javascript" (si possono usare indifferentemente lettere mi-

nuscole o maiuscole). Per comodità si userà "JavaScript".
Dall'oggetto ScriptEngine, recuperato dal nome, si può accedere alle diverse funzionalità dell'engine. Ecco le principali.

5.3 REPERIRE I RISULTATI DELLO SCRIPT

Usando il metodo eval è possibile valutare una espressione (nel linguaggio a cui l'engine si riferisce) e assegnarla ad un oggetto java; per esempio:

```
Object risultato = engine.eval("3/2");
System.out.println(risultato);
```

stampa il valore 1.5. Anche un assegnamento come:

```
Object risultato = engine.eval("risultato=3/2");
System.out.println(risultato);
```

Dà il medesimo risultato. Infatti eval restituisce sempre l'ultima espressione valutata. Tant'è che anche le seguenti istruzioni restituiscono lo stesso valore:

```
Object risultato = engine.eval(
  "if (3/2>1) risultato=5;"+
  "altroTest=2;"+
  "risultato=3/2;"
   );
System.out.println(risultato);
```

Come fare per ottenere, per esempio, il valore della variabile altroTest, assegnata nell'istruzione intermedia? In questo caso si deve ricorrere ad una particolare, chiamata Bindings, che mantiene le as-

sociazioni tra variabili e valori create all'interno dello script; infatti:

```
Bindings bind = engine.getBindings(
  ScriptContext.ENGINE_SCOPE);
System.out.println( bind.get("altroTest") );
```

Stampa il valore "2.0". Si noti come getBindings ha un parametro; ENGINE_SCOPE dice di restituire gli attributi specifici dell'engine. L'altro valore ammesso è GLOBAL_SCOPE e permette di restituire gli attributi globali a tutti gli engine.

5.4 PARAMETRIZZARE LO SCRIPT

I bindings non permettono solo di leggere i valori assegnati alle variabili nello script, ma permettono anche di settarli a partire dalle classi Java. Per esempio, il seguente algoritmo, scritto usando JavaScript, verifica se un indirizzo email è corretto (usando le espressioni regolari):

```
String jsCheckEmail =
  "if(value.match(/\\b(^(\\S+@).+((\\..{2,4}))$)\\b/gi))"+
  " ris=true;"+
  "else ris=false;";
```

Ecco come settare l'indirizzo email con i bindings e leggere il risultato della verifica:

Attenzione

Si noti che gli apici backslash ("\") non possono essere scritti da soli, in quanto sono dei caratteri speciali in Java. È necessario inserire un ulteriore backslash affinché Java lo interpreti come un effettivo carattere "\". Pertanto la stringa JavaScript "/\b(^(\S+@).+((\..{2,4}))$)\b/gi" deve essere scritta "/\\b(^(\\S+@).+((\\..{2,4}))$)\\b/gi".

```
bind.put("value", "ivanvenuti@yahoo.");
st.engine.eval(jsCheckEmail);
System.out.println( bind.get("ris"));
```

In questo caso la stampa dà False; passando un indirizzo corretto restituirà True.

5.5 FUNZIONI JAVASCRIPT

A ben vedere la quasi totalità delle funzioni JavaScript scritte nelle pagine DHTML sono nella forma di funzioni. Per far sì che da Java si possa accedere ad una di queste funzioni, dopo averne istanziato correttamente i parametri, è necessario ricorrere ad una nuova interfaccia: Invocable.
Ecco come riscrivere lo script precedente come funzione (come di solito si trova nelle librerie JavaScript per il Web):

```
String jsCheckEmailFunc =
  "function checkEmail(value) { "+
  "  if(value.match(/\\b(^(\\S+@).+((\\..{2,4}))$)\\b/gi))"+
  "    ris=true;"+
  "  else ris=false;"+
  "  return ris;"+
  " }";
```

Per prima cosa è necessario “far conoscere” all'engine la presenza della funzione; per farlo basta valutarla con l'usuale funzione eval; poi sull'oggetto engine si deve eseguire il cast ad Invocable; essa è un'interfaccia che possiede il metodo invokeFunction per l'esecuzione di una funzione (con gli opportuni parametri passati come array di Object):

```
engine.eval(jsCheckEmailFunc);
```

```
Invocable metodo = (Invocable) engine;
System.out.println(
  metodo.invokeFunction(
    "checkEmail",
    new Object[]{ "ivanvenuti@yahoo.it" }
  )
);
```

5.6 USARE OGGETTI/CLASSI JAVA

Una cosa estremamente utile per quanto riguarda l'engine Rhino inserito nel JDK 6 è che la sintassi JavaScript è stata estesa affinché, all'interno di uno script, ci si possa riferire anche a classi Java. Infatti è possibile istanziare un qualsiasi oggetto Java referenziandolo e creandolo proprio come si farebbe normalmente:

```
String dataSistema="data = new java.util.Date();";
System.out.println(st.engine.eval(dataSistema));
```

Non solo, è possibile dichiarare i package utilizzati usando "importPackage(nome.del.package)"; però attenzione al fatto che se esistono dei metodi/classi JavaScript omonime, l'uso dell'import può portare a risultati inattesi. Infatti eseguendo lo script:

```
String dataSistema="importPakage(java.util);"+
  "data = new Date();";
System.out.println(st.engine.eval(dataSistema));
```

ci si accorge che "new Date()" non invoca il metodo Java, ma instanzia una nuova data usando l'oggetto nativo JavaScript! Si noti come all'interno degli script, l'uso di oggetti Java non prevede né eventuali cast né la dichiarazione dei tipi delle variabili.

5.7 COMPILARE GLI SCRIPT

Com'è noto uno script è "interpretato" quando, ogni volta che esso viene eseguito, viene passato ad un interprete che ne esegue, passo passo, le istruzioni convertendole in una forma eseguibile per la piattaforma in cui viene eseguito. Uno script potrebbe anche essere compilato, se l'engine del linguaggio prevede un simile compilatore. In questo caso la traduzione da script a forma eseguibile viene fatta una volta soltanto (la prima) poi ogni eventuale nuova esecuzione non deve prevedere alcuna interpretazione. È evidente che questo può portare a vantaggi significativi di performance.
L'engine per Rhino, presente in Java 6, prevede la compilazione degli script. Per farlo, si può fare un cast dell'engine ad un oggetto di tipo Compilable (che è un'interfaccia) e invocare su tale oggetto il metodo compile che restituisce un oggetto di tipo CompiledScript; ecco un esempio:

```
String jsCheckEmail =
 "if(value.match(/\\b(^(\\S+@).+((\\..{2,4}))$)\\b/gi))"+
 "  ris=true;"+
 "else ris=false;";
CompiledScript compilato =
 ((Compilable) engine).compile(jsCheckEmail);
```

Come prima si possono usare sia i binding che il metodo eval (quest'ultimo ora senza parametri: ciò che deve essere valutato è al suo interno!):

```
bind.put("value", "ivanvenuti@yahoo.it");
System.out.println( compilato.eval() );
```

5.8 E SE CI SONO ERRORI?

Purtroppo nello scrivere del codice è fin troppo facile inserire degli

errori sintattici. Questo è ancor più vero se non si ha a disposizione un IDE o qualche altro strumento che, per lo meno, aiuti ad evidenziare le istruzioni sintatticamente scorrette. Nel caso alla funzione eval arrivi uno script sintatticamente non valido, si ha un'eccezione. Eccone un esempio (ovviamente l'errore varia in base all'errore riscontrato):

```
javax.script.ScriptException:
                              sun.org.mozilla.javascript.internal.EvaluatorException:
 missing ; before statement (<Unknown source>#1) in <Unknown source>
                                                                    at line number 1
   at com.sun.script.javascript.RhinoScriptEngine.
                                              eval (RhinoScriptEngine.java:110)
   at com.sun.script.javascript.RhinoScriptEngine.eval
                                                   (RhinoScriptEngine.java:124)
   at javax.script.AbstractScriptEngine.eval (AbstractScriptEngine.java:247)
   at it.ioprogrammo.java6.scripting.ScriptingTest.main
                                                       ( ScriptingTest.java:60)
```

Come c'è da aspettarsi un eventuale errore viene riportato durante l'esecuzione del metodo eval su uno script da interpretate, durante la compilazione (metodo compile) se questa viene usata.

5.9 USARE JRUNSCRIPT

Tra i tool a disposizione nel JDK 6.0 c'è jrunscript, un interprete a linea di comando che permette l'esecuzione di JavaScript. Tale JavaScript è lo stesso utilizzabile all'interno dei programmi Java (quindi è possibile ricorrere agli importPackage per far uso degli oggetti standard Java). Ecco, per esempio, come creare una finestra grafica:

```
js> importPackage(javax.swing)
js> frame = new JFrame("Un esempio di JFrame")
```

```
js> frame.setVisible(true)
```

Si potrà notare la creazione di una finestra grafica (in realtà appare ben poco: giusto la barra con il titolo e i pulsanti per iconizzarla, espanderla e chiuderla, **Figura 5.1**).
Proviamo ad estendere l'esempio; di seguito alle istruzioni prece-

Figura 5.1: La finestra appena creata da jrunscript

denti, scrivere:

```
js> importPackage(java.awt.event)
js> button = new JButton("Chiudimi")
js> listener = new ActionListener{ actionPerformed: function()
                                        { frame.dispose() } }
js> button.addActionListener( listener )
js> frame.add(button)
js> frame.pack()
```

Si potrà notare come l'ultima istruzione farà sì che la finestra contenga anche il nuovo pulsante (Figura 5.2); premendo il pulsante la finestra verrà chiusa. Per uscire da jrunscript basta premere i pulsanti [Ctrl]+[c]. Scriptpad: un esempio avanzato
Nella distribuzione standard di Java 6 è presente, nella cartella, \jdk6\sample\scripting\scriptpad, un esempio di uso avanzato della

Figura 5.2: La finestra con il nuovo pulsante

Sul Web

Alla pagine http://scripting.dev.java.net è possibile trovare le informazioni aggiornate sulle tecnologie di scripting rese disponibili per Java. Nel prossimo futuro ci si aspetta il supporto di linguaggi come Ruby, Python, PHP e molti altri.

API per lo scripting. Viene realizzato un editor, del tutto simile a Notepad per Windows, per la gestione degli script JavaScript (infatti l'editor permette l'apertura, modifica, salvataggio ed esecuzione di script!). Purtroppo l'esempio non viene fornito compilato. Per farlo è necessario installare Apache Ant (http://ant.apache.org/, Figura 5.3) e, dopo essersi posizionato sulla cartella scriptpad\, eseguire il comando:

```
>ant
```

Il risultato della compilazione è la generazione del file \scriptpad\build\scriptpad.jar. È possibile eseguire l'esempio con:

```
>java -jar .\build\scriptpad.jar
```

Figura 5.3: Home page di Apache Ant

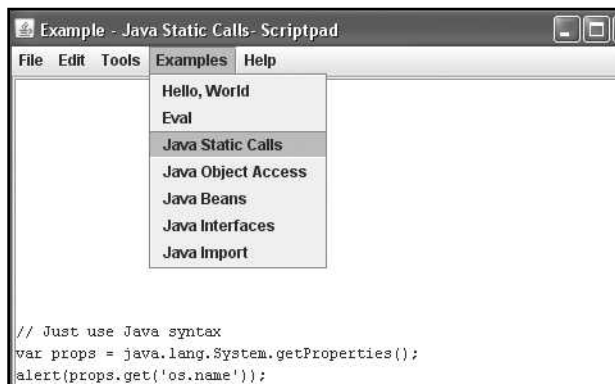



Figura 5.4: L'applicazione di esempio scriptpad

In **Figura 5.4** si vede un esempio di sua esecuzione; in particolare dal menu "Examples" è stato scelto uno degli esempi predefiniti: "Java Static Calls". Poi l'esempio è stato eseguito usando il menu "Tools > Run".

JAVA MANAGEMENT EXTENSIONS

La tecnologia JMX (Java Management eXtensions) non è una novità in Java 6. Però in questa versione numerose migliorie sono state fatte alla tecnologia per il management delle applicazioni. Vediamone le principali, dopo aver brevemente introdotto gli scopi originari della specifica. La specifica JMX illustrata è la 1.4. Come Reference Implementation è considerato il JDK 6 stesso (il JDK 5 è stata la prima versione a includere supporto nativo per l'architettura JMX).

6.1 MONITORING E MANAGEMENT DELLE RISORSE

Con il termine di monitoring si intendono strumenti e tecnologie atte ad osservare il comportamento (consumo e allocazione di risorse, tempo di esecuzione e così via) di una determinata risorsa. Per management invece si intende la possibilità di variare la sua configurazione sia in termini assoluti (nuovi valori) che in seguito a determinati eventi (come eccezioni, esaurimento etc. etc.). Questi due aspetti sono cruciali in ambienti di produzione sia per le risorse hardware (si pensi ai dischi fissi, alle porter di rete o alla memoria) che alle risorse software (sistema operativo, JVM, Web Server e applicazioni). Avere un'unica modalità per il managment e il monitoring è auspicabile; ecco perché in Java è stata introdotta la tecnologia JMX. Essa offre sia opportune API per esporre le funzionalità dei programmi sia un'infrastruttura per accedere e gestire le risorse che usano queste API.

6.2 MBEAN

Qualunque risorsa, per poter essere esposta attraverso l'architettura JMX, deve possedere un oggetto, chiamato MBean (ma-

naged bean), che (in qualche modo) ne legge gli attributi e li espone con opportuni metodi get/set; un MBean può anche esporre dei metodi (che possono anche cambiare lo stato della risorsa). In Figura 6.1 si vedono, per esempio, tre possibili MBean: uno che è un wrapper per un dispositivo hardware, uno espone le funzionalità di un software (come può essere Tomcat) e un terzo MBean generico. Questo livello viene chiamato "Instrumentation Level".

Instrumentation Level

Figura 6.1: "Instrumentation level": un esempio di risorse esposte da MBean

In concreto un MBean non è né più né meno di un Java Bean che implementa un'interfaccia che ha il suo stesso nome ma con il suffisso MBean; per esempio, ecco un possibile MBean che espone all'esterno la temperatura rilevata da un sensore (misurato in millisecondi a partire da una data di riferimento, della rilevazione):

```
public interface SensoreMBean{
  public double gradi();
  public long tempoRilevazione();
}
public class Sensore implements SensoreMBean{
  public double gradi(){
    return ((new java.util.Date()).getTime() % 330) / 10.0;

```

```
  }

  public long tempoRilevazione(){
    return (new java.util.Date()).getTime();
  }
}
```

Questo è, molto semplicemente, un MBean valido! Purtroppo ha un problema di fondo: per forza di cose chi vuol leggere una temperatura e il tempo in cui è stata rilevata dovrà fare due invocazioni distinte ai due metodi. Ma chi assicura che, per esempio, dopo aver letto i gradi, lo stato della classe cambi e quando legge il tempo di rilevazione questo non è quello riferito alla temperatura letta in precedenza? Una possibile soluzione a questo problema è definire una classe "wrapper" del tipo:

```
public class Temperatura{
  public double gradi;
  public long tempoRilevazione;
}

E modificare la classe MBean in questo modo:

public interface SensoreTemperaturaMBean{
  public Temperatura getTemperatura();
}

public class Sensore implements SensoreTemperaturaMBean{
  public Temperatura getTemperatura(){
    return new Temperatura();
  }

}
```

6.3 JMX AGENT

Un MBean da solo non è né più né meno di una classe Java. Però la specifica JMX prevede che possa esistere un MBean Server su cui si possono registrare i diversi MBean creati. Esistono poi degli Agent che forniscono servizi sugli MBean. MBean Server e Agent, nel loro complesso sono chiamati anche JMX Agent (**Figura 6.2**).

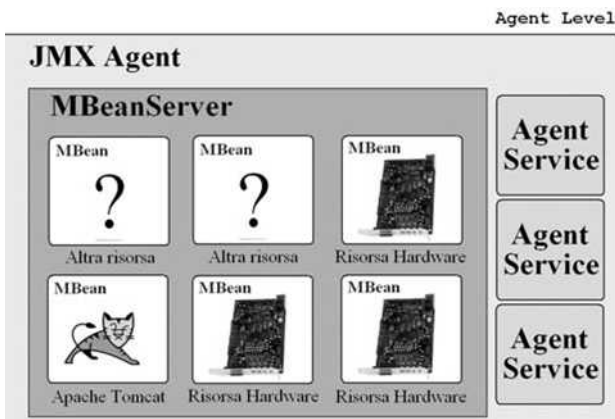



Figura 6.2: JMX Agent (un MBeanServer e uno o più agent)

In Java esiste un MBeanServer di default per la piattaforma; esso è recuperabile con:

```
MBeanServer platfMBS =
  ManagementFactory.getPlatformMBeanServer();
```

Su questo MBean Server è necessario registrare gli MBean da esporre; in questa classe, per esempio, vengono registrati i due MBean appena introdotti (quello con la classe Wrapper e quello senza) e il programma si mette in attesa "infinita" per permettere agli altri client di accedere ai due oggetti registrati:

```
public class TestMBean {

 public static final void main(String[] args) throws Exception{
   javax.management.MBeanServer platfMBS =
      getPlatformMBeanServer();
   javax.management.ObjectName nomeSensore =
      new javax.management.ObjectName(
        "it.ioprogrammo.java6.jmx:type=Sensore"
      );
   javax.management.ObjectName nomeSensoreTemperatura =
      new javax.management.ObjectName(
        "it.ioprogrammo.java6.jmx:type=SensoreTemperatura"
      );
   platfMBS.registerMBean(
     new Sensore(), nomeSensore);
   platfMBS.registerMBean(
     new SensoreTemperatura(), nomeSensoreTemperatura);
   Thread.sleep(Long.MAX_VALUE);
 }
}
```

In questo caso la registrazione di SensoreTemperatura fallisce! Il motivo è che il tipo restituito da getTemperatura() non è tra quelli standard di Java. Vediamo perché questo non può essere "ricostruito" da chi fa uso de server MBean…

6.4 SERVIZI DISTRIBUITI

Un JMX Agent, per poter essere usato, ha bisogno di un ulteriore strato software: protocol adapters e connectors. I primi (protocol adapters) sono degli ascoltatori di messaggi costruiti rispettando uno specifico protocollo (un esempio può essere http, ma anche ftp ed SMTP). Essi si trovano all'interno di un JMX Agent

e ne permettono l'uso a chiunque riesca a connettersi ad esso con il protocollo specificato (la sua fruizione può essere vincolata sia dal suo essere reso "visibile" o meno, sia da eventuali credenziali richieste; queste credenziali, ovviamente, sono anch'esse specifiche del protocollo usato). I connectors hanno lo stesso fine (mettere in comunicazione dei client remoti con il JMX Agent) ma richiedono due componenti software: uno che sta, come prima, all'interno del JMX Agent, uno sta nel client. In questo senso i due componenti "mascherano" il protocollo usato e sono essi stessi gli attori che si occupano della comunicazione (e di eventuali aspetti legati alla sicurezza). Lo schema logico risultante è mostrato in **Figura 6.3**.

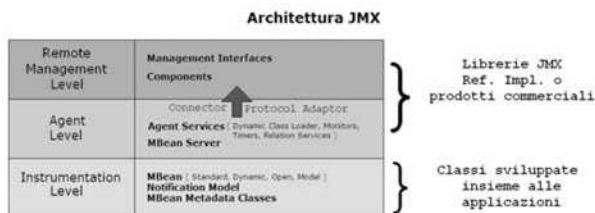



Figura 6.3: Servizi distribuiti: connectors e protocol adapters

Una situazione come quella precedente, in cui la classe Temperatura è definita solo a livello di MBeanServer, non potrebbe permettere l'accesso alla classe a meno che la classe stessa (Temperatura) non sia disponibile ai diversi client. Inoltre non è possibile fare né questa assunzione né usare un MBean standard; al più si può usare un Dynamic MBean la cui definizione è alquanto complessa. Prima di Java 6 non c'era altre alternative. Ora, invece …

MXBean

Gli MXBean nascono proprio per risolvere, in maniera semplice, i problemi appena incontrati. Benché in Java 5 esistessero già degli MXBean (presenti nel package java.lang.management) erano definiti solo all'interno delle librerie di java e non era possibile crearne di nuovi; con Java 6 è possibile definirne di nuovi. Mantenendo il vincolo sui tipi di dato referenziati, una qualsiasi classe Java può essere un MXBean purché implementi un'apposita interfaccia. Affinché l'interfaccia sia valida per definire MXBean può avere una di queste caratteristiche:

```
Avere come suffisso MXBean (per esempio TestPerMXBean)
Avere l'annotazione @MXBean
```

Si noti che la classe che implementa l'interfaccia non deve, come accadeva per gli MBean, essere chiamata come l'interfaccia stessa ma senza il suffisso (TestPer, riprendendo l'esempio sopracitato); infatti basta che implementi l'interfaccia e può avere un nome qualsiasi. Ecco un esempio di MXBean:

```
public interface SensoreTemperaturaNewMXBean {
    public Temperatura getTemperatura();
}

public class SensoreTemperaturaNew implements
                                    SensoreTemperaturaNewMXBean {
    private Temperatura t;

    public SensoreTemperaturaNew(Temperatura t){
        this.t = t;
    }
    public Temperatura getTemperatura(){
       double gradi = ((new java.util.Date()).getTime() % 330) / 10.0;
       long tempoRilevazione = (new java.util.Date()).getTime();
```

```
        t = new Temperatura(gradi, tempoRilevazione);
        return t;
    }
}
```

Fin qui nulla di strano; ecco come deve essere la classe Temperatura:

```
public class Temperatura {
    private double gradi;
    private long tempo;

    public double getGradi(){
        return gradi;
    }

    public long getTempo(){
        return tempo;
    }

    @java.beans.ConstructorProperties({ "gradi", "tempo"})
    public Temperatura(double gradi, long tempo){
        this.gradi = gradi;
        this.tempo = tempo;
    }
}
```

Anch'essa è un JavaBean come gli altri ma, e deve essere così!, il costruttore possiede un'annotazione ConstructorProperties: grazie a tale notazione si inseriscono importanti metadati sui nomi dei parametri che verranno utilizzati.
Ora si può riscrivere la registrazione dell'MXBean in questo modo:

```
javax.management.ObjectName nomeSensoreTemperatura =
   new javax.management.ObjectName(
    "it.ioprogrammo.java6.jmx:type=SensoreTemperaturaNew"
   );
platfMBS.registerMBean((Object) new SensoreTemperaturaNew(
   new Temperatura(3.2, (new java.util.Date().getTime()))
     , nomeSensoreTemperatura);
```

6.5 JAVA VM

La Virtual Machine di Java può essere essa stessa monitorata e configurata via JMX.
Non solo: con Java 6 esiste una cartella, sotto la home del JDK, chiamata /demo/management, dove sono presenti applicazioni d'esempio per il monitoring della JVM. In Figura 6.4, per esempio, il risultato dell'esecuzione di:

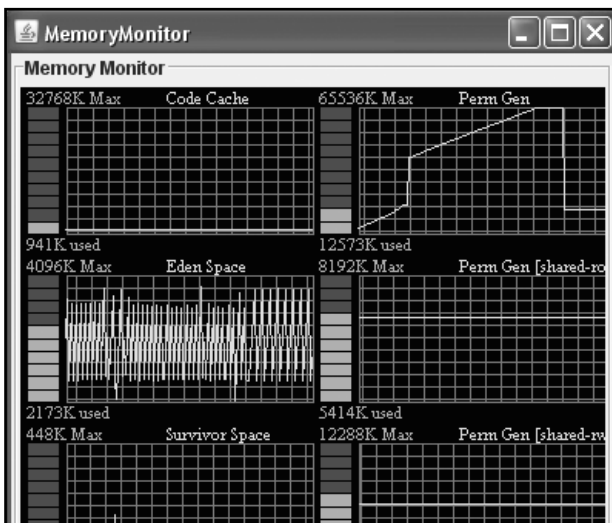



Figura 6.4: L'applicazione MemoryMonitor in azione

```
…\demo\management\MemoryMonitor>java -jar MemoryMonitor.jar
```

Numerosi tool sono inclusi in Java 6 per il debug delle applicazioni. Alcune, come JConsole, erano già presenti in Java 5 ma hanno subito notevoli evoluzioni e miglioramenti. Vediamo i più significativi.

6.6 JCONSOLE

JConsole è un tool grafico per monitorare le risorse esposte via JMX. Può automaticamente agganciarsi ad una qualsiasi Java VM in esecuzione sulla macchina locale se questa è una JVM 1.6; per le JVM versione 1.5 è necessario che queste siano state mandate in esecuzione con l'opzione: -Dcom.sun.management.jmxremote. Teoricamente anche JVM precedenti possono essere monitorate, purché dispongano di un proprio JMX Agent. È anche possibile connettersi a server remoti. Per conoscere le opzioni a disposizione si può far eseguire il comando:

```
jconsole -help
```

apparirà una finestra grafica con le opzioni disponibili (**Figura 6.5**). Quello che interessa è mandare in esecuzione la classe TestMBean e mettersi “in ascolto” con JConsole. Per farlo esegui-

```
JConsole: Output
Usage: jconsole [ -interval=n ] [ -notile ] [ -pluginpath <path> ] [ -version ] [ connection ... ]

  -interval    Set the update interval to n seconds (default is 4 seconds)
  -notile      Do not tile windows initially (for two or more connections)
  -pluginpath  Specify the path that jconsole uses to look up the plugins
  -version     Print program version

  connection = pid || host:port || JMX URL (service:jmx:<protocol>://...)
  pid          The process id of a target process
  host         A remote host name or IP address
  port         The port number for the remote connection

  -J           Specify the input arguments to the Java virtual machine
               on which jconsole is running
```

Figura 6.5: Le opzioni per jconsole

re dapprima la classe e poi JConsole. Appariranno una serie di connessioni disponibili: ognuna rappresenta una JVM in esecuzione sulla macchina a cui ci si può agganciare.
Facendo alcune prove (purtroppo sono riportati solo i PID dei processi attivi, per cui da essi è difficile capire da essi qual è la JVM associata all'esempio). Ci si accorgerà di monitorare la JVM corretta quando si vedrà, nel tab MBean, la gerarchia it.ioprogrammo.java6.jmx. Provando a navigarne il contenuto si può esservare che ci sono entrambi i bean registrati: Sensore e SensoreTemperaturaNew (Figura 6.6). Per quanto riguarda il bean

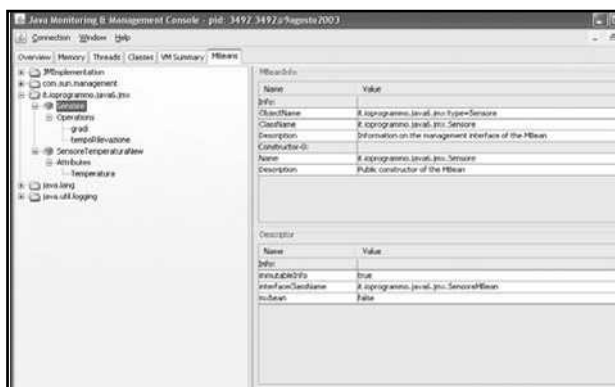



Figura 6.6: I due bean acceduti da JConsole

Sensore, ci sono due operazioni, ognuna con il metodo getter dell'attributo esposto. Facendo clic sul pulsante che ha lo stesso nome del metodo getter, viene visualizzata una finestra con il valore dell'attributo corrente (**Figura 6.7**). Diverso il caso del bean SensoreTemperaturaNew. Infatti esso ha una proprietà di un tipo particolare: CompositeDataSupport. Facendo doppio clic su tale tipo, si apre un semplice navigatore delle proprietà (in questo caso vengono lette insieme, in quanto viene letto l'oggetto wrapper!). Il tutto è illustrato in **figura 6.8.**

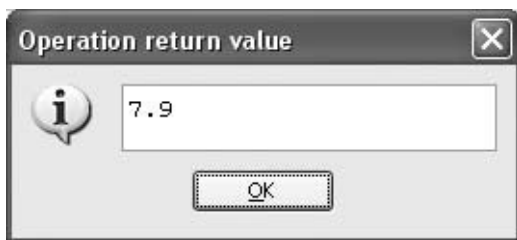



Figura 6.7: Valore della proprietà gradi

6.7 UN PLUGIN PER JCONSOLE

JConsole è estendibile con appositi plugin. Infatti un plugin può essere pacchettizzato in un archivio JSR; eseguendo jconsole con il parametro -pluginpath e specificando la posizione del file JAR, il plugin contenuto viene mandato in esecuzione. Per esempio, posizionarsi nella home di installazione del JDK 6 ed

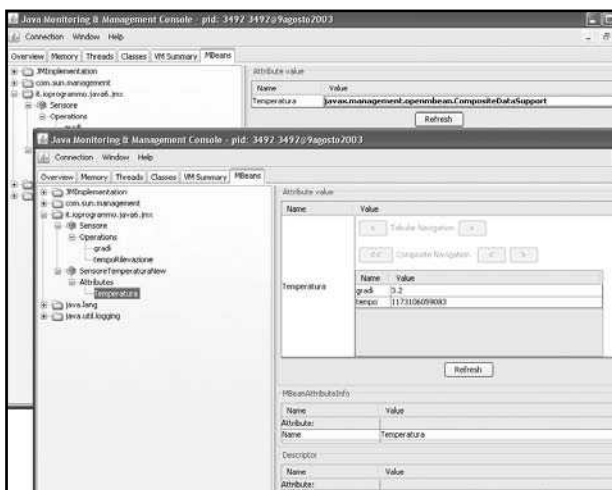



Figura 6.8: Proprietà temperatura

eseguire il seguente comando:

```
jconsole -pluginpath ./demo/scripting/jconsole-plugin/
                                        jconsole-plugin.jar
```

Questo fa sì che il plugin di esempio (di cui sono disponibili i sorgenti) venga caricato. Nella finestra grafica di JConsole si può notare un nuovo tab chimato "Script Shell" da dove è possibile eseguire dei comandi in JavaScript. Ecco, per esempio, come referenziare l'MBean "Runtime" e stampare la versione della JVM (**Figura 6.9**):

```
rt=mbean("java.lang:type=Runtime")
rt.VmVersion
```

Esiste anche un file che definisce delle funzioni di utilità; tale file è in jdk1.6.0\demo\scripting\jconsole-plugin\src\resources e si chiama jconsole.js. Lo si può aprire che un editor testuale, copiarlo e incollarlo nella nuova shell di comandi. Ora si ha a disposizione tutta una serie di funzioni utili ad interagire con jconsole e gli MBean. Per conoscere i comandi a disposizione si può digitare:

```
js>help()
```

Per conoscere le informazioni associate all'MBean interrogato in precdenza si può usare:

Figura 6.9: Lo Script Shell e un esempio d'uso

```
js>mbeanInfo("java.lang:type=Runtime")
```

Esiste anche un altro plug in per JConsole, con relativi sorgenti, nella distribuzione standard del JDK 1.6; quest'altro plugin si trova in /jdk1.6.0/demo/management/JTop.
Un plugin personalizzato?
Per realizzare un proprio plugin per JConsole è possibile estendere la classe com.sun.tools.jconsole.JConsolePlugin e implementare i due metodi getTab e newSwingWorker. Il primo restituisce la lista dei tab da aggiungere, il secondo restituisce una classe che estende SwingWorker che è responsabile dell'aggiornamento grafico del plugin:

```
public Map<String, JPanel> getTabs() {
}

public SwingWorker<?, ?> newSwingWorker() {
}
```

6.8 TOOL PER IL DEBUG

Il JDK 6.0 ha una ricca dotazione di tool per agevolare il debug delle applicazioni. In questo contesto per debug non si intende tanto il processo di esecuzione e monitor delle applicazioni du-

Attenzione
Se non viene trovata la classe com.sun.tools.jconsole.JConsolePlugin significa che non è presente nel classpath il file jconsole.jar che si trova nella cartella lib/ dell'installazione del JDK (e non nel JRE!). Se si usa NetBeans fare clic con il pulsante destro del mouse sul progetto e scegliere "Properties". Nelle Categorie (a sinistra) scegliere "Libraries": nella parte destra, nel tab "Compile", premere "Add JAR" e selezionare la libreria lib/jconsole.jar.

rante il processo di sviluppo (in questi casi qualsiasi IDE avanzato offre strumenti molto più adatti allo scopo) ma quanto il processo di identificazione di problemi sulle applicazioni in produzione. Per una panoramica dei diversi strumenti (molti dei quali legati proprio alla tecnologia JMX) si può far riferimento alla documentazione ufficiale alla pagina http://java.sun.com/javase/6/webnotes/trouble/.

6.9 PER IL FUTURO?

JMX è un'architettura in continua evoluzione e il cui uso è pressoché totale per tutti i progetti di una certa complessità (sia per quanto riguarda le tecnologie lato server come Tomcat, JBoss, WebSphere che altri tool di sviluppo come Eclipse). L'evoluzione della piattaforma è tesa sempre più alla sua semplificazione e alla presenza di nuovi tool di supporto. È consigliabile l'uso di JMX in qualsiasi progetto "critico", dove il monitoring e la configurazione (anche dinamica) sono aspetti cruciali e non tralasciabili. Per il futuro si potrà usare nuovi tipi di MBean, come i "Virtual MBean" (http://blogs.sun.com/nickstephen/entry/jmx_extending_the_mbeanserver_to).

Sul web

Un'ottima fonte per gli aggiornamenti sulla tecnologia JMX e sull'uso dei componenti esistenti è il blog raggiungibile alla pagina http://weblogs.java.net/blog/emcmanus/.

ALTRE NOVITÀ

In quest'ultimo capitolo vengono elencate, in una panoramica, altre novità introdotte da Java 6 e che non rientrano nelle macro categorie descritte in precedenza. In conclusione si vedrà la nuova licenza open source della piattaforma e cosa (e quando) ci si può aspettare per la prossima release.

7.1 API DEL COMPILATORE

Attraverso il nuovo package javax.tools vengono esposte classi che permetto l'uso del compilatore da parte dei programmi Java, grazie ad una serie di nuove API che ne comandano il comportamento.
Ecco un semplice esempio che reperisce il compilatore di sistema, grazie ad una getSystemJavaCompiler sulla classe ToolProvider, poi esegue la stampa delle versioni supportate e, infine, esegue la compilazione vera e propria grazie al comando run.
Tale compilazione usa gli usuali canali (standard input, output ed error) e compila quanto viene passato come argomento alla classe stessa (è possibile passare sia un array di stringhe che un numero qualunque di stringhe grazie al numero variabile di parametri di run).
Nell'esempio si passa gli argomenti passati al metodo main direttamente al metodo run:

```
package it.ioprogrammo.java6.compiler;

```

Attenzione
È possibile passare al metodo run gran parte delle opzioni documentate per l'interprete; una eccezione è rappresentata dall'opzione -J: essa permette di specificare le opzioni per la Java Virtual Machine; affinché abbiano effetto non è possibile passarle al metodo run, ma dovrebbero essere specificate alla JVM usata per eseguire il metodo run!

```
import javax.tools.*;

public class TestCompiler {

 public static final void main(String[] args)
 throws Exception{

   JavaCompiler jcompiler =
    ToolProvider.getSystemJavaCompiler();
   java.util.Set<javax.lang.model.SourceVersion>
    sver = jcompiler.getSourceVersions();
   // Stampa le versioni supportate
   for(javax.lang.model.SourceVersion source: sver){
     System.out.println(source.name());
   }
   // compila, usando gli argomenti della classe
   jcompiler.run(
     System.in,
     System.out,
     System.err,
     args
   );
 }
}
```

In **Figura 7.1** si può osservare il comportamento dell'invocazione della nuova classe senza parametri:

```
java it.ioprogrammo.java6.compiler.TestCompiler
```

oltre a stampare i nomi simbolici delle versioni supportate, stampa la lista delle opzioni disponibili per il compilatore; proprio come se si fosse invocato javac senza parametri!

```
C:\Documents and Settings\TdT\Java6EsempiLibro\build\classes>java it.ioprogrammo
.java6.compiler.TestCompiler
RELEASE_3
RELEASE_4
RELEASE_5
RELEASE_6
Usage: javac <options> <source files>
where possible options include:
  -g                         Generate all debugging info
  -g:none                    Generate no debugging info
  -g:{lines,vars,source}     Generate only some debugging info
  -nowarn                    Generate no warnings
  -verbose                   Output messages about what the compiler is doing
  -deprecation               Output source locations where deprecated APIs are u
sed
  -classpath <path>          Specify where to find user class files and annotati
on processors
  -cp <path>                 Specify where to find user class files and annotati
on processors
  -sourcepath <path>         Specify where to find input source files
  -bootclasspath <path>      Override location of bootstrap class files
  -extdirs <dirs>            Override location of installed extensions
  -endorseddirs <dirs>       Override location of endorsed standards path
  -proc:{none,only}          Control whether annotation processing and/or compil
ation is done.
  -processor <class1>[,<class2>,<class3>...]Names of the annotation processors t
o run; bypasses default discovery process
  -processorpath <path>      Specify where to find annotation processors
  -d <directory>             Specify where to place generated class files
```

Figura 7.1: L'invocazione della classe di test senza paramentri

Passando come parametri i nomi di classi java da compilare, la classe ne effettua la compilazione.

Attenzione

È interessante notare che il reperimento del compilatore del sistema può avvenire solo se l'ambiente Java è installato correttamente; pertanto è necessario settare opportunamente le variabili di sistema JAVA_HOME, CLASSPATH e PATH. Se questo non avviene il metodo jcompiler.getSourceVersions restituirà null.

7.2 CONOSCERE I PARAMETRI DI RETE

La classe java.net.NetworkInterfaces è stata estesa per poter reperire tutta una serie aggiuntiva di informazioni sulle interfacce di rete presenti; ecco una semplice routine che ne mostra l'uso:

```
public void printParametriRete(int livello,
    java.util.Enumeration<java.net.NetworkInterface> networkInterfaces)
    throws SocketException{
```

```
 if (networkInterfaces==null)
    return;
 while( networkInterfaces.hasMoreElements()){
   java.net.NetworkInterface network =
     networkInterfaces.nextElement();
   System.out.println( "("+livello+") "+ network.getName() +
     "( "+network.getDisplayName()+")");
   System.out.println( "            isUp : "+
     network.isUp());
   System.out.println( "      isLoopback : "+
     network.isLoopback());
   System.out.println( "       isVirtual : "+
     network.isVirtual());
   System.out.println( "  isPointToPoint : "+
     network.isPointToPoint());
   System.out.println( " supportsMulticast : "+
     network.supportsMulticast());
   printParametriRete( livello+1, network.getSubInterfaces());
 }

}
```

Essa può essere invocata con:

```
obj.printParametriRete(
 1,
 java.net.NetworkInterface.getNetworkInterfaces()
);
```

Numerosi altri metodi permettono di conoscere, per esempio, indirizzi e sottomaschere per ciascuna interfaccia.

7.3 QUANTI BYTE SU DISCO?

Finalmente è possibile sapere la composizione di ciascuna partizione. Infatti sono stati introdotti i metodi getTotalSpace, getFreeSpace e getUsableSpace nella classe java.io.File. Essi restituiscono, rispettivamente, lo spazio totale, libero ed utilizzabile della partizione. La differenza tra libero ed usabile è che il primo indica i byte non utilizzati, ma che possono essere anche dedicati esclusivamente ad altri utenti (grazie alla restrizione sui diritti di accesso alle varie cartelle, per esempio) mentre per usabile si intende il numero di byte che effettivamente la partizione permette di scrivere. Ecco come, a partire dalla lista delle partizioni disponibili, stampare tali valori:

```
public void printPartizioni(){
  java.io.File[] listaRoot = java.io.File.listRoots();
  for(java.io.File file: listaRoot){
    System.out.println(file.getPath()+
        " liberi " + file.getFreeSpace() +
        " usabili " + file.getUsableSpace() +
        " byte su " + file.getTotalSpace());
  }
}
```

7.4 DIRITTI SUI FILE

La classe java.io.File ha metodi che permettono il test dei diritti su file; in particolare canRead() e canWrite() restituiscono un valore booleano che indica, rispettivamente, se si hanno i diritti di lettura e scrittura sul file/cartella a cui l'oggetto File si riferisce. In Java 6 è disponibile un terzo metodo: canExecute(); esso indica se si possiedono i diritti di esecuzione sul file/cartella specificati. Java 6 introduce, per gli oggetti File, nuovi metodi per settare tali diritti; in particolare:

- **setExecutable(boolean executable); setExecutable(boolean executable, boolean ownerOnly)**: permette di rendere (o meno) eseguibile un file; il secondo parametro (del secondo metodo) indica se il diritto va abilitato/disabilitato solo per l'utente che lo ha creato o per tutti (sempreché il file system lo permetta; se non distingue gli utenti allora il secondo parametro viene ignorato ed è equivalente all'invocazione del primo; se invece il file system distingue gli utenti, il primo metodo equivale al secondo a cui viene passato True come secondo parametro, ovvero si applica al solo creatore del file); se tutto va a buon fine i metodi restituiscono True;
- **setReadable(boolean readable) , setReadable(boolean readable, boolean ownerOnly)**: stesso comportamento dei metodi precedenti, ma si riferiscono al diritto di lettura;
- **setWritable(boolean writable), setWritable(boolean writable, boolean ownerOnly)**: gestiscono il diritto di scrittura sul file.

I diversi metodi possono sollevare un'eccezione di tipo SecurityException nel caso non si abbia i diritti di scrittura sul file. Ecco un metodo che riceve, come argomento, una stringa unix like per i diritti sui file: essa viene composta come una stringa di 6 caratteri dove le diverse posizioni specificano, rispettivamente, i diritti di lettura owner, scrittura owner, esecuzione owner, lettura altri utenti, scrittura altri utenti ed esecuzione altri utenti. Se i diritti vengono assegnati c'è , ripettivamente "r", "w" o "x"; se i diritti vengono negati c'è un singolo trattino: "-" (per semplicità si potrebbe trattare in maniera equivalente un qualsiasi altro carattere). Secondo tale convenzione la stringa "rwxrwx" significa che tutti hanno tutti i diritti sul file; "rw-rw-" invece indica solo i diritti di lettura e scrittura (per qualsiasi utente); "rw----" intende i diritti di lettura e scrittura per il solo owner. Si noti che una stringa come "---rwx" non ha molto senso, in quanto (per l'implementazione Java!) i diritti assegnabili agli altri utenti

devono essere assegnati anche all'owner (in pratica la nostra procedura potrebbe trattare questo tipo di stringhe come “------”). Ecco una possibile implementazione:

```
public void setRights(File file, String rights) throws Exception{
  if (rights==null || rights.length()!=6)
    throw new Exception("Numero parametri errati!");
  file.setReadable( rights.charAt(0)=='r',
    rights.charAt(0)!=rights.charAt(3));
  file.setWritable( rights.charAt(1)=='w',
    rights.charAt(1)!=rights.charAt(4));
  file.setExecutable( rights.charAt(2)=='x',
    rights.charAt(2)!=rights.charAt(5));
}
```

Ecco un possibile modo di invocare il metodo:

```
altro.setRights(new File("/home/test/test.txt"), "rwx-w-");
```

7.5 UN INTERPRETE AGGIORNATO

L'interprete Java ora può accettare caratteri speciali per l'inclusione di librerie, senza doverle più specificare una ad una:

```
java -cp lib/* nomeDelFile
```

Inoltre è stato migliorato il verificatore di codice incorporato (per i dettagli si veda JSR 202, https://jdk.dev.java.net/verifier.html). Infatti è stata introdotta un'architettura nuova, chiamata “split verification”, che deriva da Java ME: grazie a questa architettura si hanno sia migliori performance che una semplificazione architetturale (che ha permesso una maggior verificabilità della sicurezza intrinseca).
Sono state introdotte nuove opzioni per l'interprete. Per quanto ri-

guarda le opzioni disponibili con la modalità -XX, si può far riferimento alla pagina web: http://www.md.pp.ru/~eu/jdk6options.html.

7.6 PERFORMANCE

In genere è stato ottenuto un notevole aumento delle performance. Per un'analisi dettagliata si può far riferimento all'articolo "Java 6 Leads Out of the Box Server Performance" reperibile alla pagina http://blogs.sun.com/dagastine/entry/java_6_leads_out_of (Figura 7.2). Alla pagina http://weblogs.java.net/blog/campbell/archive/2005/07/strcrazier_perf.html vengono segnalati miglioramenti per la performance delle primitive OpenGL (l'articolo è "STR-Crazier: Performance Improvements in Mustang").

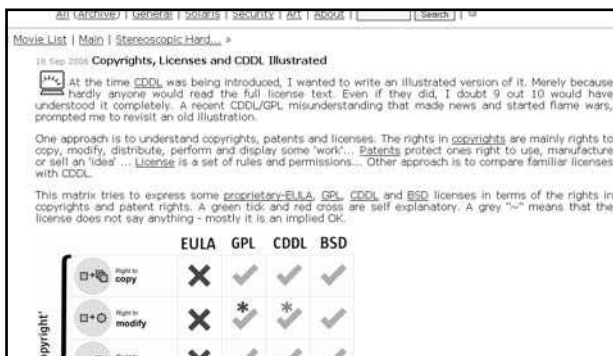



Figura 7.2: Analisi delle prestazioni server di diversi JDK

7.7 LA NUOVA LICENZA: GPL VERSIONE 2

I vari ambienti di Java 6 sono stati rilasciati sotto licenza GPL versione 2 (per i dettagli sulla licenza si può far riferimento alla pagina http://www.gnu.org/copyleft/gpl.html, mentre per i dettagli dell'ap-

plicazione di tale licenza alle diverse piattaforme si può far riferimento alla pagina http://www.sun.com/software/opensource/java/).
In realtà usare la licenza imporrebbe a chiunque usi Java che, a sua volta, le applicazioni sviluppate debbano essere rilasciate con licenza GNU. Ovviamente questa sarebbe del tutto improponibile visto l'uso attuale di Java in moltissime applicazioni commerciali. Sun ha specificato, pertanto, una eccezione, chiamata CLASSPATH Exception: in pratica solo eventuali modifiche agli ambienti di sviluppo e al linguaggio dovranno, a loro volta, essere rilasciate con licenza GPL. Chi usa Java solo come piattaforma di sviluppo di applicazioni può adottare una qualsiasi licenza per la loro distribuzione.
La comunità open source ha accolto con entusiasmo questo annuncio di Sun. Aldilà di questioni filosofiche, è indubbio che questo annuncio porterà nuove forze per contribuire allo sviluppo di Java; basti pensare che attualmente esistono ben due progetti che miravano alla creazione di un ambiente Java alternativo, completamente open source; il primo è un progetto di GNU, http://www.gnu.org/software/classpath/, il secondo di Apache: http://harmony.apache.org/. Mi aspetto che, a breve, queste forze confluiranno nel progetto di Sun dando ulteriore vigore e slancio al suo sviluppo. Chi avesse dei dubbi sui diversi tipi di licenze "libere", può far riferimento alla pagina http://blogs.sun.com/chandan/entry/copyrights_licenses_and_cddl_illustrated per una loro comparazione.
Ma quali migliorie e innovazioni aspettarsi per la prossima release?

7.7 ASPETTANDO JAVA 7

Concludendo si può osservare che Java 6 ha, rispetto a Java 5, molti miglioramenti (soprattutto in termini di
performance) ma non moltissime evoluzioni a livello di linguaggio.
Java 5 ha visto il rilascio dopo quasi 3 anni di sviluppo. Java 6, invece, è stata rilasciata circa un anno dopo Java 5. Questo è in linea con le linee guida del management della Sun, che vorrebbe rilascia-

re spesso nuove release del linguaggio senza aspettare tante nuove caratteristiche com'è stato per Java 5 (a tal proposito si veda l'intervista alla pagina http://java.sun.com/developer/technicalArticles/Interviews/hamilton_qa2.html). Secondo tali linee guida c'è da aspettarsi che Java 7 (chiamata "Dolphin", delfino) vedrà la luce più o meno agli inizi del 2008. Non c'è ancora una JSR ufficiale che ne tracci i contenuti; però è possibile dedurre alcune novità guardando un po' sul web, soprattutto curiosando tra i blog degli sviluppatori e responsabili del progetto. In particolare è interessante la pagina http://blogs.sun.com/dannycoward/entry/channeling_java_se_7. Ecco, in ordine sparso, cosa ci si può aspettare…

7.8 MODULI

Particolare interesse suscita la JSR 277, "Java Module System", per l'introduzione del concetto di modulo. Un modulo nasce dall'esigenza di superare le limitazioni dei file JAR, gli attuali archivi per la distribuzione del codice. Essi infatti non permettono un benché minimo controllo sulle versioni e rappresentano un problema nel caso di applicazioni complesse che necessitano di molti e diversificati file (di cui poi vengono dimenticati il motivo e le dipendenze con i sorgenti!).
Non solo: chi sviluppa con IDE sa che ognuno di essi ha un proprio modo di gestire le precedenze dei file inclusi, gli application server un altro e così via. Se qualche classe viene distribuita su più JAR diversi questo può rappresentare un serio problema di incompatibilità difficilmente identificabile. Al momento è disponibile solo una versione preliminare della specifica (early draft) ed è prevista anche un'integrazione con la specifica JSR-277, "Improved Modularity Support in the Java Programming Language". Alla pagina http://www.bejug.org/confluenceBeJUG/display/PARLEYS/JSR-277%20Java%20Module%20System una interessante presentazione (purtroppo solo in inglese!) della tecnologia.

7.9 NUOVA API DI PERSISTENZA

In Java 7 potrebbe esserci l'attesa JSR 220, “Enterprise JavaBeans 3.0”, una nuova (ed enormemente semplificata) architettura per la persistenza dei dati. Da sempre l'accesso ai DBMS rappresenta uno degli aspetti cruciali di gran parte delle applicazioni. La sua introduzione sarebbe un punto interessante e fondamentale per adottare la nuova release.

7.10 SWING E JAVABEAN

La specifica JSR 296, “Swing Application Framework”, ha lo scopo di semplificare lo sviluppo di applicazioni Swing tentando di fattorizzare tutto il codice ripetitivo che si deve scrivere nella creazione di interfacce grafiche (una presentazione degli obiettivi e la descrizione di un prototipo sono reperibili alla pagina http://weblogs.java.net/blog/hansmuller/archive/jsr296-presentation.pdf). Questa fattorizazione dovrebbe portare alla creazione di un framework unico e semplice da usare. Utili anche le specifiche JSR 295, “Beans Binding”, e JSR 303, “Bean Validation”, che, rispettivamente, permettono di mantenere la sincronia tra bean diversi e validare (attraverso delle regole o un altro meccanismo standard) i contenuti dei bean. Per un esempio di Beans Binding si veda il post
http://weblogs.java.net/blog/zixle/archive/2007/02/update_on_beans.html (pur essendo ancora lontana la definizione della specifica, è un ottimo esempio per comprenderne l'uso).

7.11 ULTERIORI LINGUAGGI DI SCRIPTING

Si è avuto modo di apprezzare, in Java 6, la presenza di un engine di scripting per JavaScript. Ci si aspetta che altri linguaggi siano introdotti nella release successiva. In particolare destano in-

teresse e curiosità linguaggi come Ruby e Python (con le rispettive implementazioni Java JRuby e Jython). Questo sarebbe agevolato dall'introduzione della JSR 292, "Supporting Dynamically Typed Languages on the Java Platform", che permetterebbe l'uso di nuovi bytecode a livello di JVM per l'invocazione di metodi in cui i tipi di dato dei parametri e del risultato non è nota al compile type, ma viene risolta solo al runtime al momento stesso dell'invocazione.

7.12 JMX ANCORA… PIÙ POTENTE!

Anche sull'architettura JMX sono in corso interessanti sviluppo, a riprova della validità dell'architettura e alla sua importanza strategica. In questo contesto potrebbero essere introdotte le specifiche JSR 255, (la versione 2.0 di JMX!) e JSR 262, Web Services Connector for Java Management Extensions (JMXTM) Agents.

7.13 CLOSURES?

Ultimamente si fa un gran parlare di closures e della possibilità che anche Java le supporti. Forse la versione 7 introdurrà anche questa novità? Per ora ci sono dei tentativi di specifiche, come si può vedere alla pagina http://www.javac.info/ ("Closures for the Java Programming Language").

7.14 DOWNLOAD DI JAVA 7? SI PUÒ!

Una novità rispetto alle versioni precedenti (ma non per Java 6!) è che fin d'ora è possibile scaricare la versione in lavorazione di Java 7 dal sito https://jdk7.dev.java.net/. Circa ogni settimana

viene aggiornata la versione scaricabile. E' ovvio che tale versione è ancora lontana dall'essere stabile ed effettivamente usabile in un ambiente di produzione, ma è un ottimo modo per "toccare con mano" l'evolversi della piattaforma.

i libri di

LAVORARE CON JAVA 6

Autore: Ivan Venuti

EDITORE
Edizioni Master S.p.A.
Sede di Milano: Via Ariberto, 24 - 20123 Milano
Sede di Rende: C.da Lecco, zona ind. - 87036 Rende (CS)

Realizzazione grafica:
Cromatika Srl
C.da Lecco, zona ind. - 87036 Rende (CS)

Art Director: Paolo Cristiano
Responsabile grafico di progetto: Salvatore Vuono
Coordinatore tecnico: Giancarlo Sicilia
Illustrazioni: Tonino Intieri
Impaginazione elettronica: Francesco Cospite

Servizio Clienti
☎ **Tel. 02 831212 - Fax 02 83121206**
@ **e-mail: customercare@edmaster.it**

Stampa: Grafica Editoriale Printing - Bologna

Finito di stampare nel mese di Aprile 2007